Democracy Enhancing Technologies:
Toward deployable and incoercible E2E elections

by

Jeremy Clark

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

End-to-end verifiable election systems (E2E systems) provide a provably correct tally while maintaining the secrecy of each voter's ballot, even if the voter is complicit in demonstrating how they voted. Providing voter incoercibility is one of the main challenges of designing E2E systems, particularly in the case of internet voting. A second challenge is building deployable, human-voteable E2E systems that conform to election laws and conventions. This dissertation examines deployability, coercion-resistance, and their intersection in election systems. In the course of this study, we introduce three new election systems, (Scantegrity, Eperio, and Selections), report on two real-world elections using E2E systems (Punchscan and Scantegrity), and study incoercibility issues in one deployed system (Punchscan). In addition, we propose and study new practical primitives for random beacons, secret printing, and panic passwords. These are tools that can be used in an election to, respectively, generate publicly verifiable random numbers, distribute the printing of secrets between non-colluding printers, and to covertly signal duress during authentication. While developed to solve specific problems in deployable and incoercible E2E systems, these techniques may be of independent interest.

Supervisor

- Urs Hengartner

Committee & Examiners

- Ian Goldberg
- Alfred Menezes
- Steve Schneider
- Douglas R. Stinson

Acknowledgements

I would first like to thank my advisor, Urs Hengartner, for his guidance through my PhD. Urs was always keen on our research, motivating, and accommodating. I admire his sensibility toward solving real-world problems. Much of this dissertation would not have been the same without his thorough examination of the protocols, threat models, and their presentation.

I would like to thank the rest of my committee, Ian Goldberg, Alfred Menezes, Steve Schneider and Doug Stinson for their corrections, suggestions, and prompts for clarification when it was most certainly needed.

I had the pleasure to work with David Chaum on three of his projects: Punchscan, Scantegrity, and a forthcoming project. I learned a lot from David, both technically and on a range of other subjects. I also enjoyed the collaboration with the rest of the team: Rick Carback, Aleks Essex, Stefan Popovenuic, Ron Rivest, Emily Shen, Alan Sherman, Poorvi Vora, and Filip Zagórski. A special acknowledgement to Rick, who once suggested using panic passwords for internet voting. That suggestion motivated much of Part II of this dissertation.

The CrySP lab was an excellent incubator for ideas and friendship. In particular, I would like to thank Aniket Kate and Greg Zaverucha. Both provided as much knowledge and assistance as they did entertainment and friendship. Thank you to the rest of lab as well and my friends in the broader Waterloo graduate community (sorry, too many people to name).

I'd like to especially thank one person who intersects both Scantegrity and CrySP, as well as almost my whole academic career: Aleks Essex. Aleks is always the first person I bounce ideas off of and has been a great friend through many of my major life events.

The least likely person to ever read this is Clint Mansell but I feel indebted to his musical genius, and hope some of his creativity transferred to all the words I wrote as his scores swelled around me.

I'd like to thank my family; in particular, my parents Marvin and Sharon and my brother Jonathan. As well, I'd like to thank new family: Subhash and family, Sindhu and family, and Shaan and Shunker.

Last but not least, thank you to my wife Neha for her support, welcome distractions, tolerance, and encouragement. It is impossible to describe all direct and indirect ways she helped.

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through an Alexander Graham Bell Canada Graduate Scholarship.

Contents

List of Tables

List of Figures

Chapter 1

Introductory Remarks

The rise of democracy among the independent states of the world is one of the most important recent global trends. Diamond marks the beginning of the "third-wave of global democratization" with Portugal's transition to democracy in 1974 [Dia08]. Prior to this event, he reports that a mere 27% of the independent states in the world were democracies. By 1999, the number had climbed to 63% and has remained roughly steady since. While liberal democrats should find this global view encouraging, a domestic look will not necessarily be so optimistic, especially in the United States where Stewart finds that voter distrust in the electoral process remains, even after a relatively uneventful 2008 election [Ste08].

One source of diminished voter confidence is in electronic election technology. Electronic vote-counting methods have become increasingly prominent. By 2006, less than 1% of the United States voting-age population still voted by hand-counted paper ballot, with over 88% of the votes being tallied either by optical-scan, or some other form of electronic ballot tabulation [Ele06]. Gronke and Hicks find that respondents in the United States assigned a mean score of 4.57 out of 5 in agreement with the statement: "We need to make elections more secure" [Ans09]. This outranked issues such as voter access and the convenience of voting. In Germany, a 2009 Supreme Court ruling banned the use of electronic voting machines, observing that "programming errors in the software or deliberate electoral fraud committed by manipulating the software of electronic voting machines can be recognized only with difficulty" [Fed09].

The security of current electronic election technology is difficult to ascertain. Vendors use intellectual property protection laws to deter independent reviews of their systems, and on the rare occasions when reviews are authorized by state executives or the judiciary, they are often time-constrained affairs whose specific findings are subject to strict non-disclosure agreements. Yet despite this limited window of access, security experts have found no shortage of serious security flaws and vulnerabilities when permitted to look (or otherwise

obtain access). Beginning in 2004 with Kohno *et al.* [KSRW04], independent reviews have continued to uncover serious security vulnerabilities that allow, among other things, undetectable tally manipulation if an attacker gains access to the system under reasonable threat scenarios. Recent studies have found such problems in systems by the four largest vendors: ES&S [ACC+08], Diebold[1] [CFH+07, BEH+08], Sequoia[1] [BCE+07, AGH+09], and Hart-InterCivic [IRS+07, BEH+08].

The subject of this dissertation is on a radically different approach to detecting faults and fraud in elections: end-to-end verification (E2E) of election results provided by cryptographic techniques. E2E election systems allow voters to verify that their ballots are processed correctly, independent of the correct functioning of any computerized component in the system. In essence, end-to-end verifiable voting systems provide a cryptographic chain of custody on the ballots, from the start, when the votes are cast, to the end, when they are counted. They also enable the correctness of the final tally to be verified by anyone. Cryptographic voting protocols and systems have evolved since they were first proposed by Chaum in 1981 [Cha81]. Early proposals assumed voters were trusted to not try and reveal how they voted, either under coercion or for money, and assumed voters could perform computations during the vote casting process. The work of this dissertation continues the more recent trend in the literature of building voter-verifiable systems that are both incoercible and practical.

Thesis Statement. End-to-end verifiable voting systems, and their components, can be designed for real-world deployability while maintaining a strong notion of ballot secrecy and incoercibility, even in the case of internet voting.

1.1 Background

Throughout the 1980s and 1990s, many protocols for cryptographic voting were proposed, the majority based on either mix networks, originating with Chaum [Cha81], or additive homomorphic encryption, originating with Benaloh and Tuinstra [BT94]. We review these in Chapter 3. This first generation of protocols implicitly assumed each voter could reliably perform computations, like encrypting messages and verifying proofs, in order to submit her ballot.

In the 2000s, the literature moved in two directions. The first direction is toward human ballot casting protocols, which still use cryptography but allow voters to "encrypt" their ballots in the voting booth without computations. These systems are the focus of

[1]Diebold's election division was renamed Premier Election Solutions in 2007. In May 2010, Dominion Voting Systems acquired Premier and that June it acquired Sequoia.

Part I. The second direction is toward remote voting protocols, especially internet voting, where voters can perform computations. However, unlike in the first generation of cryptographic voting protocols, the computations are not assumed to be reliable as their personal computers may be infected with malware, and voters are no longer isolated in a voting booth—they could cast votes in the presence of a coercer or vote buyer, or give up their credentials allowing the adversary to cast the ballot. Maintaining incoercibility in internet voting systems is the focus of Part II.

1.2 Contributions

While the research into end-to-end verifiable systems has been substantial, our work provides a number of new contributions toward the development of secure and deployable E2E systems. We briefly summarize them here.

Punchscan Case Study. Punchscan is a paper-based E2E election system. We present a case study of Punchscan's first use in a binding election. The election was held in March 2007 at the University of Ottawa for several offices within the university's graduate student association. This case study presents a walkthrough of the election and offers discussion as to how the voters and poll workers responded to the Punchscan system, with implications to the deployability of E2E systems in general.

Scantegrity. We introduce Scantegrity, a practical E2E enhancement for optical scan voting systems that achieves increased election integrity through the use of confirmation codes printed on the ballots in invisible ink. Voters mark ballots just as in conventional optical scan but using a special pen that develops the invisible ink. The voters can optionally copy down the codes and check that the system received their vote correctly, however the codes do not prove which candidate the voter cast her ballot for, maintaining incoercibility. Unlike previous systems in the literature, Scantegrity is a cryptographic add-on that still allows standard electronic counting of the ballots, as well as manual recounting. This allows Scantegrity to be deployed within existing procedures and with existing technology.

Scantegrity Case Study. On November 3, 2009, 1728 voters in Takoma Park, Maryland, cast ballots for the mayor and city council members using the Scantegrity voting system—the first time any paper-based E2E election system has been used in a binding governmental election. We describe the various efforts that went into the election—including the improved design and implementation of the voting system, streamlined procedures, agreements with the city, and assessments of the experiences of voters and poll workers.

We also provide useful observations and lessons that may assist others in deploying E2E systems.

Coercion Contracts. We consider coercion contracts in voting systems with end-to-end (E2E) verifiability. Coercion contracts are a set of instructions that an adversary can dictate to a voter, either through duress or by offering payment, that increase the probability of a compliant voter constructing a vote for the adversary's preferred candidate.[2] Using Punchscan as a representative deployed E2E system, we place the attacks in game-theoretic terms and study the effectiveness of three proposed contracts from the literature. We offer a definition of optimality for contracts, provide an algorithm for generating optimal contracts, and show that as the number of candidates increases, the adversary's advantage through the use of contracts decreases. We also consider the use of contracts in a heterogeneous population of voters and for financially constrained adversaries. This work furthers our understanding of incoercibility and illustrates it with an E2E system that has seen real-world use.

Random Beacons. In E2E election systems, random challenges are sometimes used to verify that the tally was computed correctly. In the elections using Punchscan and Scantegrity, randomness was drawn from stock market data to generate these challenges. Assuming market fluctuations are unpredictable to some degree, these challenges cannot be known in advance. However the degree to which these movements are unpredictable is not known to be sufficient for generating a fair challenge. We use tools from computational finance to provide an estimate of the amount of entropy in the closing price of a stock. We estimate that for each of the 30 stocks in the Dow Jones industrial average, the entropy is between 6 and 9 bits per trading day. We then propose a straight forward protocol for regularly publishing verifiable 128-bit random seeds with entropy harvested over time from stock prices. These beacons can be used as challenges directly, or as a seed to a deterministic pseudorandom generator for creating larger challenges. We believe this is a practical, real-world approach to providing universal verification of fair random challenges.

Eperio. In terms of execution time and software complexity, the technical requirements for conducting an E2E election audit can be prohibitive. In an effort to reduce these requirements, we present Eperio: a provably secure construction for providing a tally that can be efficiently verified using only a small set of primitives. We show how common place utilities, like the use of file encryption, can further simplify the verification process for

[2]While contracts that force a voter to cast a ballot for a random candidate could be considered a coercion contract, we concentrate on contracts that bias the receipt toward a selected candidate, for any candidate on the ballot.

election auditors. Compared to other proposed proof-verification methods for end-to-end election audits, Eperio lowers the technical requirements in terms of execution time, data download times, and code size. Auditing can be completed using the built-in functions of a spreadsheet, making Eperio the first end-to-end system to not require special-purpose verification software. Eperio can interface with ballots from Prêt à Voter, Punchscan, and, with a limitation, Scantegrity.

Secure Printing. In paper-based E2E elections, ballots are printed by a trusted entity. We consider the problem of how to print a human-readable message or image on a piece of paper, without the printing agents or devices learning its contents. We examine the problem in two settings: with a trusted dealer who knows the secret and with a two-party protocol that allows two non-colluding printers to obliviously generate the secret and print it while never learning it. The protocols allow the printers to print a randomly selected element from a set or a random permutation of elements in a set. When the elements are alphanumeric strings, we offer an optimization for representing them. By obliviously printing codes or shuffled lists of candidates, these protocols could form the basis for eliminating the trusted printer in paper-based E2E election systems like Prêt à Voter, Punchscan, and Scantegrity.

Panic Passwords. Panic passwords allow a user to signal duress during authentication, for example, when being coerced while voting online. We show that the well-known model of giving a user two passwords, a 'regular' and a 'panic' password, is susceptible to attack, and is secure only within a very narrow threat model. We expand this threat model significantly, making explicit assumptions and tracking four parameters. We also introduce several new panic password systems to address new categories of scenarios, including the `5-Dictionary` scheme which can be deployed in a wide variety of real-world applications, including internet voting.

Selections. We present Selections, a new cryptographic voting protocol that is end-to-end verifiable and suitable for internet voting. After a one-time in-person registration, voters can cast ballots in an arbitrary number of elections. We say a system provides over-the-shoulder coercion-resistance if a voter can undetectably avoid complying with an adversary that is present during the vote casting process. Our system is the first in the literature to offer this property without the voter having to anticipate coercion and precompute values. Instead, a voter can employ a panic password. We believe this more practical for real-world voters.

1.3 Organization

In the next chapter (Chapter 2), we present some background on voting technology, its use in Canada, and the cryptographic primitives necessary to understanding the remainder of this work. The rest of the work is divided into two parts: Part I concentrates on in-person voting systems, while Part II focuses on internet voting systems.

Part I begins with a survey of existing work on in-person voting systems, in Chapter 3. We then consider two E2E systems, Punchscan and Scantegrity, in Chapter 4. The remaining work in this chapter touches on one of these two systems in some way.

Using Punchscan, in Chapter 5 we employ a game-theoretic analysis of an adversary's ability to coerce voters in the original definition of the system. We then examine an open issue with both Punchscan and Scantegrity: where do the publicly verifiable random numbers come from for the audit? In Chapter 6, we describe a random beacon constructed from the closing prices of financial stocks. We then consider if the cryptographic audit trail of Punchscan and Scantegrity could be made simpler and more efficient to verify. In Chapter 7, we describe a small and efficient audit trail that can be verified using spreadsheets. Finally in Chapter 8, we consider some preliminary results for further improving in-person E2E voting systems like Punchscan and Scantegrity.

Part II begins with Chapter 9, a survey of existing work on coercion-resistant, E2E internet election systems. In Chapter 10, we examine the use of panic passwords and create new schemes that could be implemented in an election system. In Chapter 11, we do exactly that: we describe Selections, a coercion resistant Internet voting system based on panic passwords.

We offer some concluding remarks in Chapter 12.

Chapter 2

Preliminaries

Consider two traditional election systems. The first is the public vote, where voters write down their votes on a publicly posted list. This system allows voters and observers to compute their own tally to independently verify the asserted result. We refer to this property as integrity. The second system is the ballot box, where voters deposit secret votes into a sealed box and go home. This system allows the voters' selections to be unlinkable to their identities. We refer to this property as ballot secrecy.

First, note that neither of these systems has both integrity and secrecy. A public vote allows anyone to observe each voter's selection, and a ballot box election provides no assurance that votes have not been modified, removed, or added to the asserted outcome. The first goal of applying cryptographic techniques to voting is to derive a voting scheme that has *both* integrity and ballot secrecy.

Next, note that neither of the two traditional schemes is necessarily infallible in the property that they do hold. The public list may be modified without a voter noticing the change. The privacy of a ballot box election could be compromised if voters made unique marks on their ballots, voted in a booth with a hidden camera, *etc.*. End-to-end verifiable (E2E) election systems use these systems as a baseline. They aim to provide *as much* integrity as a public vote and *as much* secrecy as a ballot box. Some E2E schemes may attempt to address problems beyond this, but we do not consider it necessary.

2.1 Election Technologies

Throughout the dissertation, we occasionally make reference to existing election technologies. We provide a quick summary of notable technology here.

- **Paper Ballots and Ballot Box**. At the beginning of the election, a ballot box is demonstrated to any observers to be empty and is sealed. Voters mark a paper ballot in a private booth and cast it into the box. After the election, the box is unsealed and tallied in the presence of any observers.
- **Optical Scan**. The paper ballot system is augmented with an optical scanner. Voters mark a paper ballot, typically by filling in an oval or joining two segments of an arrow. Voters cast the ballot by feeding it into a scanner. Typically, the scanner automatically deposits the scanned ballot into the ballot box. Electronic votes are recorded on a memory card in the scanner.
- **HAVA-compliant Optical Scan**. In the United States, the Help America Vote Act (HAVA)[1] specifies that voters should be warned before casting a ballot with contests that are undervoted or overvoted. In such an error condition, the ballot is reverse fed out of the scanner so that the voter can correct the ballot or be issued a new one. If the voter wishes to proceed, an election official can override the error, typically with a physical key.
- **Punchcard**. Punchcards are a predecessor to optical scan. Ballots are marked by punching a hole in the mark position for a candidate. Voters cast the ballot by feeding it into a scanner. A partially punched hole may leave a hanging chad, which can be inadvertently swung back into place when fed into the scanner, causing the mark not to register.
- **Direct Electronic Recording (DRE) Machine**. Voters are issued an access code or card that they type in or insert into a computer terminal. The voter selects candidates using physical buttons on the machine or through a touchscreen. Once the ballot is complete, the voter uses a button to cast their vote. The vote is recorded on a memory card in the machine.
- **Voter-Verified Paper Audit Trail (VVPAT)**. DRE machines can be augmented with paper receipts. When voters have completed their ballot, the machine prints out a summary of the selections on a receipt that is displayed to the voter under glass. The voter verifies that it is consistent and then casts the ballot. The vote is recorded on the memory card and the receipt tape is advanced to no longer display the vote.
- **Lever Machines**. Lever machines are a predecessor to DRE machines. Voters select candidates by toggling physical switches and cast their vote by pulling a lever. Mechanical counters in the back of the machine are incremented to maintain a tally.
- **Ballot Marking Device (BMD)**. Voters are issued a paper ballot that they place in the BMD terminal. Using buttons, they enter their candidate selections. When the ballot is complete, the BMD marks the paper ballot for the voter. The voter then casts the ballot into an optical scanner or into the ballot box, as the case may

[1]Public Law 107-252, The Help America Vote Act of 2004

be. BMDs are primarily used by voters with disabilities that prevent them from marking a paper ballot, such as visual impairment or motor disabilities. Both DREs and BMDs provide audio instructions.

- **Mail-in Ballots**. Absentee voters can fill out a paper ballot and return it by mail. The ballot is typically enclosed in a double envelope. The outer envelope contains the voter information, precinct, and signature, and is used to verify eligibility. For eligible ballots, the inner envelope is removed from the outer envelope and added to the collection of other absentee ballots. The inner envelopes do not contain any identifying information. When tallied, the ballots are removed from inner envelopes and fed into a scanner or hand-counted.
- **Internet Voting**. Absentee voters visit a website or install a program on their computer to capture their vote. The software transmits the electronic ballot over the internet to the election authority. In certain circumstances (*e.g.,* overseas military), voters may be required to attend a remote polling place and use a dedicated kiosk, instead of their own personal computer.

2.1.1 Elections in Canada

In Canada, Federal elections are conducted with a paper ballot system. Voters with disabilities are permitted to vote with assistance from another person. Polling places also offer a tool that ballots can be placed into, creating a physical template for the ballot with Braille markings for each mark position. This can be of assistance to visually impaired voters. Absentee voters can vote by mail.

To personally verify the integrity of the tally, one is limited to observing a single polling place and would need to devote a full day to observation. In the case of a recount, this tally will be superseded by a final tally that is generally closed to the public, and tamper-resistant tape is the only guarantee that the contents of ballot boxes have not been modified between tallies. The goal of E2E election systems is to improve this situation by allowing personal verification of the final tally for all precincts at a time of the verifier's choice after the election.

Municipal elections in Canada use a variety of technologies. Larger municipalities often use optical scanners or DREs. Several Canadian municipalities have also begun to use internet voting. Without end-to-end verifiability, such electronic forms of vote capture and tallying provide little assurance of their accuracy. Typically the code is not available for independent validation and even if it were, it can be difficult to verify its correctness. Furthermore, voters and observers cannot perform attestation procedures to ensure the proper code is running on the machines. We emphasize that our position is not that testing, certification, and attestation is fundamentally unsolvable *within a reasonable threat model*, however the current state of the election technologies and procedures is far from achieving

it. On the other hand, cryptographic techniques can side-step these issues and provide reasonable assurance of the results under the assumption that the election technology is untrustworthy.

2.2 Requirements

In this section, we outline the requirements for end-to-end verifiable (E2E) election systems. We break them into three groups. First we list the core technical requirements, which relate to integrity and ballot secrecy. We then consider some additional technical requirements that may arguably be sub-requirements of the core requirements, but we list them separately to emphasize the advanced nature of the requirements. Finally we list some requirements relating to the deployability of the system, which may or may not be considered in a particular E2E design.

Too many papers in the literature make reference to some subset of these properties to cite them all, however we include some citations to papers that originate or substantially address a specific requirement.

2.2.1 Core Requirements

Integrity. Integrity (correctness, verifiability) is the requirement that an incorrect tally can be detected with high probability. A correct tally should be accepted (completeness), while an incorrect tally should be detectable (soundness). The decision may be available to voters who participated in the election and retained information (voter verifiability), by anyone examining a public transcript of the election (universal verifiability), or a combination. The tally may be incorrect due to errors, fraud, or attacks.

Ballot Secrecy. Ballot secrecy (privacy) is the requirement that voters' selections remain private with high probability. This requirement is weak in the sense that while an adversary can consult the public transcript of the election and any information retained by the voter, he cannot corrupt the voters themselves. He functions as a passive eavesdropper on all tappable channels. Note that the tally itself can leak information about voters' selections (*e.g.,* an election with only one voter), therefore ballot secrecy can be stated more accurately as the requirement that voters' selections remain *as* private as when only a tally is output.

Coercion-Resistance. Coercion-resistance (receipt-freeness) [BT94, Oka97, JCJ05, MN06, KTV10b] is the requirement that voters' selections remain secret with high probability even

when the voters themselves may be corrupted. A coerced voter may deliberately take actions to provide evidence of which candidate she selected or more general properties of her vote (*e.g.,* she abstained, voted for a random candidate, *etc.*). An adversary should not be able to decide if the coerced voter was actually corrupted or if the voter is deceiving the adversary (generally, deception is made possible through an untappable channel (voting booth) that the adversary cannot eavesdrop on).

The coercion-resistance requirement subsumes ballot secrecy. Early systems in the literature consider only the first two core requirements, while most recent systems consider all three. In Chapter 5, we examine one case of an election system that has ballot secrecy but not coercion-resistance due to a subtle, unintentional oversight in its original design.

There is some disagreement in the literature as to whether coercion-resistance is equivalent to receipt-freeness. The term receipt-freeness was proposed for in-person voting [BT94] but was not formalized until later [Oka97] (and this formalization was rarely used). Coercion resistance was proposed for internet voting [JCJ05] and considered receipt-freeness to be a weaker property. Some literature preserves this distinction, where coercion-resistance is considered the stronger property. For example, receipt-freeness may only allow the adversary access to the voter after the vote has been cast and not before [DKR06]. However recent definitions of receipt-freeness [MN06] and coercion resistance [KTV10b] both consider essentially the same types of threats. When summarizing the literature, we will use whichever term the authors chose to use.

2.2.2 Potential Requirements

In addition to the three core requirements, recent proposals in the literature have considered to varying degrees the following additional requirements:

- **Perfection**. In the core technical requirements, each requirement was met with high probability. Many systems are designed to instead have unconditional integrity, where correctness is not based on any intractability assumptions. They could alternatively have perfect secrecy (everlasting privacy) [Cha88, CFSY96, MN06], or even have both (unconditional security, information-theoretic security) [Yao82, PW92, BT08, OI10].
- **Tallying Authority**. Protocols where the voters themselves tally their own votes are called boardroom voting protocols [BF85]. This setting allows for information-theoretic security but generally does not scale well. All the E2E election systems we consider onward from here will use an untrusted tallying authority to whom voters will cast their ballots. The use of commitments and encryption will preclude these systems from holding both privacy and integrity unconditionally.
- **Eligibility**. The tally should only include votes from eligible voters, and the number of votes each voter is permitted to cast may be limited (*e.g.,* one vote per voter).

- **Independence.** If a voter cannot determine the selections of another voter, she should not be able to cast the same ballot as that voter (or a function of it) [Gen95]. Also known as simultaneity.
- **Dispute-freeness.** If the verification of some aspect of the election fails, implying an error or fraud, the voter should be able to demonstrate that it failed while maintaining ballot secrecy and coercion-resistance [KY02, KTV10a]. Also known as accountability.
- **Fairness.** A partial tally should not be available to anyone before all voters have cast their votes.
- **Robustness.** If a bounded number of participants refuse to or are unable to participate in tallying the votes, a correct tally can still be produced [CFSY96, CGS97].
- **Covert Channels**. Designs may attempt to eliminate covert channels (subliminal channels) that can be exploited to leak information or bypass attempts at anonymization. Issues here are subtle and usually specific to the assumptions made about the devices, channels, or participants in the election [KSW05, AN09, GGR09, FB09].

2.2.3 Deployability Requirements

We have examined a number of requirements that are generally related to the security of election systems. When consideration is given to the real world, a number of additional requirements may emerge. Recall that in our thesis statement, we emphasized the deployability of the contributions we will make. Therefore, these requirements are considered to greater extent in this work than in much of the literature.

- **Candidate Options**. Deployable systems should allow multiple contests per ballot, with multiple candidates in each contest (multi-candidate). Some early cryptographic systems allowed only binary votes (*i.e.,* yes or no).
- **Scoring Protocols**. If not otherwise specified, election systems are assumed to implement the first-past-the-post (FPTP) scoring protocol where voters specify a single preferred candidate, and the winning candidate must possess a plurality of votes. Systems may be required to implement alternatives such as Borda, instant-runoff voting (IRV), or single transferable vote (STV) where voters rank the candidates by preference.
- **Usability**. Voters should be able to confidently mark and cast a ballot. A system is said to be bare-handed if the voter can perform the necessary actions without a computational device [RT07].
- **Ballot Layout**. In some jurisdictions, there are legal requirements concerning how ballots are printed or displayed. For example, serial numbers may be required or not permitted, and candidates may need to be presented in a specific order (*e.g.,* alphabetically).

- **Auditability**. E2E election systems provide the ability to cryptographically audit the results of the election. While we hope this audit is eventually taken as definitive, some jurisdictions legally require the ability to perform traditional audits, such as hand counts of the paper trail (manual recounts).
- **Understandability**. While difficult to qualify, E2E election systems should make verification understandable to as many voters as possible, in particular, any aspects of the verification process that are only voter verifiable (and not universally verifiable) [DJ02, ECHA10].

2.3 Cryptographic Primitives

In this section, we describe the main cryptographic primitives and protocols that will be used in the dissertation. In particular, we concentrate on primitives we refer to in multiple chapters. If a particular primitive is only used in one chapter, we may defer to the description to the relevant chapter.

2.3.1 Mathematical Background

Discrete Logarithms. Many of our results and their related work can be implemented in a general discrete log setting, however we will describe them in the multiplicative subgroup $\mathbb{G}_q$ of $\mathbb{Z}_p^*$, where p and q are large primes and $p = 2\alpha q + 1$ for an integer α. For example, $|p| = 1024$ bits and $|q| = 160$ bits. An exception is Chapter 6, where we use a bilinear group introduced in that chapter.

We assume the computational intractability of the DDH-Problem within $\mathbb{G}_q$, which implies the intractability of the DL-Problem. We say a problem is intractable if it cannot be solved by a probabilistic polynomial time (PPT-bounded) algorithm. Let $g \in \mathbb{G}_q$ be a generator and $a, b, z \in_r \mathbb{Z}_q$ be random exponents. The DL-Problem is to compute a given $\langle g, g^a \rangle$. The DDH-Problem is to distinguish $\langle g, g^a, g^b, g^{ab} \rangle$ from $\langle g, g^a, g^b, g^z \rangle$.

Negligible, Overwhelming. We say a function f is negligible in some positive integer k if $f(k) < 1/\mathsf{poly}(k) = \mathsf{negl}(k)$. We say the probability of event A is high in k if $\mathbf{Pr}[A] \geq 1 - 1/k$ and is overwhelming in k if $\mathbf{Pr}[A] \geq 1 - \mathsf{negl}(k)$.

2.3.2 Commitments

Commitment schemes allow a sender to commit to a message while keeping it hidden from the receiver. The commitment can later be opened, but only to the original message. These

requirements are called hiding and binding respectively.

DL-Commitment. Let $g \in \mathbb{G}_q$ be a generator and $m \in \mathbb{Z}_q$ be a message. A basic discrete logarithm commitment can be implemented as $\mathsf{Com}_{\mathsf{DL}}(m) = g^m$. To open the commitment, the sender discloses m. The receiver can recompute $\mathsf{Com}_{\mathsf{DL}}(m)$ to check. Given g generates $\mathbb{G}_q$, the commitment is perfectly binding and it is computationally hiding if the DL-Problem is intractable.

Pedersen Commitment. Let $h \in \mathbb{G}_q$ be a second generator and $r \in_r \mathbb{Z}_q$ be a randomly selected exponent. Given g and h, there exists a such that $g^a = h$. Assume a is unknown to the sender or receiver; thus computing it is intractable. An alternative commitment, commonly attributed to Pedersen [Ped92],[2] is implemented as $\mathsf{Com}_{\mathsf{Ped}}(m, r) = g^m h^r (= g^{m+ar})$. To open the commitment, the sender reveals m and r, and the receiver verifies the commitment by recomputing it.

Let c be a Pedersen commitment to m and r; that is, $c = g^m h^r$. The commitment is perfectly hiding. Given c, $\forall m' \exists r'$ such that $g^{m'} h^{r'} = c$. However such an r' can only be found with knowledge of a: $r' = r + a(m - m')$. Therefore it is computationally binding when a is unknown.

Randomized DL-Commitment. Pedersen commitments are also randomized, unlike the DL-commitment. A randomized commitment that is perfectly binding and computationally hiding can be constructed as $\mathsf{Com}_{\mathsf{RDL}}(m, r) = \langle \mathsf{Com}_{\mathsf{Ped}}(m, r), \mathsf{Com}_{\mathsf{DL}}(r) \rangle = \langle g^m h^r, g^r \rangle$. Since it is perfectly binding, there is no restriction on the sender's knowledge of a (for simplicity, one could set $a = 1$).

Trapdoor Pedersen Commitment. In a Pedersen commitment, we assumed a was unknown to both the sender and receiver. However if a were known to the receiver only, then from the sender's view, it is still a binding commitment scheme. In this case, a is a trapdoor value that revokes the bindingness of the commitment. After revealing the committed value, the sender can safely learn a assuming the commitment is not to be reused. Trapdoor commitments can provide the receiver and/or sender with *a posteriori* deniability about the message committed to, and can also be used to transform a Σ-protocol into a zero-knowledge proof (both described below).

[2] It however appears in at least two earlier works [CDG87, BCC88].

Remarks. Each of these commitment schemes are malleable. For example, if Alice commits to message m, Bob can commit to a function of Alice's message (*e.g.,* $2m$ or $m+1$) without actually knowing m. If Alice opens hers first, Bob can use Alice's opening to compute the values needed to open his commitment. Thus other commitment schemes have been developed to provide non-malleability or other advanced properties (*e.g.,* equivocally, concurrent composition, *etc.*). Generally, malleability will not be a concern and if it is, Σ-protocols (described below) can ensure the committer knows the message being committed to.

2.3.3 Encryption

Encryption schemes allow a sender to transmit a message to the receiver, while keeping the message hidden from anyone else. The receiver can recover the original message using a secret key. We now describe the Elgamal encryption scheme [Elg84].

Elgamal. Let $g \in \mathbb{G}_q$ be a known generator. The receiver chooses a random exponent $sk \in_r \mathbb{Z}_q$ for his secret key, and publishes $y = g^{sk}$ as his public key. The sender selects random exponent $r \in_r \mathbb{Z}_q$ and encrypts message $m \in \mathbb{G}_q$ as $\langle c_1, c_2 \rangle = \mathsf{Enc}(m, r) = \langle g^r, my^r \rangle$. The receiver decrypts the message as $m = \mathsf{Dec}_{sk}(c_1, c_2) = c_1^{-sk} c_2$.

Elgamal ciphertexts are indistinguishable under a chosen plaintext attack: it has CPA-security [TY98].[3] This notion is formalized as a game. An adversary is permitted blackbox access to an encryption oracle for a given sk. He can query messages and receive ciphertexts. The adversary submits two messages, m_1 and m_2. The oracle randomly decides which to encrypt and sends the ciphertext. The adversary may then continue submitting messages and receiving ciphertexts. Finally, the adversary outputs a guess of which message was encrypted. For Elgamal, if an adversary can consistently succeed with a better probability than a random guess, he can translate this success into solving the DDH-problem. If the latter is assumed to be intractable, the former must be as well.

Like the commitment schemes, Elgamal is malleable. If $\langle c_1, c_2 \rangle = \mathsf{Enc}(m, r)$ and $a \in \mathbb{G}_q$, then $\langle c_1, ac_2 \rangle = \mathsf{Enc}(am, r)$ and if $b \in \mathbb{Z}_q$ then $\langle c_1^b, c_2^b \rangle = \mathsf{Enc}(m^b, r^b)$. Further, given two encryptions, $\mathsf{Enc}(m_1, r_1) \cdot \mathsf{Enc}(m_2, r_2) = \mathsf{Enc}(m_1 m_2, r_1 + r_2)$. This latter type of malleability is a multiplicative homomorphism. For election systems, additive homomorphisms are often more useful.

Exponential Elgamal. A variant of Elgamal [CGS97], often called exponential Elgamal, implements encryption as $\langle c_1, c_2 \rangle = \mathsf{Enc}_y(m, r) = \langle g^r, g^m y^r \rangle$. Note it has the same form

[3] Elgamal is also IND-CCA1 secure [Lip10] but we do not make use of this fact.

as the randomized, perfectly binding commitment introduced above. The "decryption" function is unchanged, $g^m = \mathsf{Dec}_{sk}(c_1, c_2) = c_1^{-sk} c_2$, however it does not recover m itself but rather g^m. Recovering an arbitrary m is intractable under the DL-problem, but if m is from a small message space, then m can be found with one of two approaches: all possible g^m values can be precomputed with $O(|m|)$ work, or m can be computed after g^m is known with $O(\sqrt{|m|})$ work.

Exponential Elgamal is useful because it is additively homomorphic: $\mathsf{Enc}(m_1, r_1) \cdot \mathsf{Enc}(m_2, r_2) = \mathsf{Enc}(m_1 + m_2, r_1 + r_2)$. For example, in a referendum, each voter could encrypt a 1 to vote yes or a -1 to vote no. All the ciphertexts could be multiplied together, creating a sum of all the votes. If only the sum is decrypted, the voters learn the result without learning individual votes. For a small number of voters, the final message will be small enough to be recoverable. This is only a sketch of an election system that omits many important details. We will return to it in Chapter 3.

Rerandomization. For both types of Elgamal, it is possible to take as input a ciphertext and output an encryption of the same message with different randomness, without knowledge of sk or the original r. Recall $\langle c_1, c_2 \rangle = \langle g^r, my^r \rangle$. Then $\mathsf{ReRand}(c_1, c_2, r') = \langle c_1 g^{r'}, c_2 y^{r'} \rangle$ produces the equivalent of $\langle c_1', c_2' \rangle = \mathsf{Enc}(m, r+r')$. Note that $\langle g, y, c_1'/c_1, c_2'/c_2 \rangle = \langle g, g^{sk}, g^{r'}, g^{sk \cdot r'} \rangle$ is a DDH-tuple only if m is not changed by ReRand. Thus, without r' or sk, under the DDH-problem, one cannot distinguish a rerandomized ciphertext from an encryption of a different message.

This function is often referred to as reencryption instead of rerandomization in the literature. We use the latter to clearly distinguish it from proxy reencryption, where a message encrypted under, say, Alice's key is changed to an encryption of the same message under Bob's key.

Threshold Elgamal. Threshold encryption of Elgamal [DF89, Ped91, GJKR99] allows sk to be shared among n trustees, such that for $m < n$, any $m + 1$ can recover sk or distributively decrypt a ciphertext. Distributed key generation $\mathsf{DKG}(n, m)$ generates a public key, e, and a private key share, sk_i, for each of n trustees. Elgamal encryption is unchanged. Distributed decryption, $\mathsf{DDec}_{sk_i}(c)$, on ciphertext c can be performed with $m + 1$ trustees using their shares sk_i. After running DDec, no trustee learns any other sk_i or the master sk. Proactive security can maintain the secrecy of the shares over time, both the number of shares and the threshold can be adjusted without a dealer, and a more complex access structure than m-out-of-n can be created.

Other Encryption Schemes. It is possible to construct additive homomorphic encryption schemes in settings other than discrete log groups. For example, Pallier encryp-

tion [Pai99, DJ01] allows direct decryption of the message without searching the message space, however the decryption function itself is less efficient than exponential Elgamal. Back in the discrete log setting, Cramer-Shoup [CS98] is an efficient construction of a non-malleable encryption scheme. It is also possible to construct encryption schemes that are non-malleable with respect to the message but still rerandomizable [PR07]. These may have other drawbacks, however, such as how efficient it is to prove facts about encrypted messages or to create a variant with threshold decryption.

2.3.4 Σ-Protocols

A Σ-protocol is a three-move, interactive proof of a statement run between a prover $\mathcal{P}$ and an honest verifier $\mathcal{V}$, where the verifier only learns whether the statement is correct [CDM00, HL10]. It is perhaps best illustrated with an example.

Given a description of $\mathbb{G}_q$, a generator $g \in \mathbb{G}_q$, and group element $y \in \mathbb{G}_q$, the prover claims to know a w such that $y = g^w$. We call $\langle q, g, y \rangle$ the common input and w the witness. $\mathcal{P}$ could naively prove knowledge of w by providing $\mathcal{V}$ with w, and $\mathcal{V}$ could verify that $y = g^w$. However $\mathcal{V}$ learns much more than only whether the statement (that $\mathcal{P}$ knows such a w) is correct.

A Σ-protocol for the knowledge of a discrete logarithm is given by Schnorr [Sch91]. $\mathcal{P}$ selects $a \in_r \mathbb{Z}_q$ and computes $b = g^a$. $\mathcal{P}$ sends b to $\mathcal{V}$. $\mathcal{V}$ selects a random bitstring of length $t \leq \log_2 q$: $c \in_r \{0,1\}^t$. $\mathcal{V}$ sends c to $\mathcal{P}$. $\mathcal{P}$ computes $d = cw + a \pmod{q}$ and sends d to $\mathcal{V}$. We call $\langle b, c, d \rangle$ a transcript. $\mathcal{V}$ accepts the transcript iff $g^d = by^c$.

Properties. Σ-protocols are three move protocols where $\mathcal{P}$ sends a value to $\mathcal{V}$, $\mathcal{V}$ sends a random challenge back, and $\mathcal{P}$ sends a final value. They have these properties:

- **Completeness**. If $\mathcal{P}$ has a valid witness and both parties follow the protocol, an accepting transcript will always be produced.
- **Special Soundness**. Let $\langle b, c, d \rangle$ and $\langle b, c', d' \rangle$ be two accepting transcripts, where b is the same and $c \neq c'$. It is possible to efficiently extract w from such a pair (*e.g.,* $w = (d - d')(c - c')^{-1}$ for the Schnorr protocol). The existence of such an efficient witness extractor can be used to prove that if $\mathcal{P}$ can produce an accepting transcript, $\mathcal{P}$ must know w with overwhelming probability $1 - 2^{-t}$.
- **Simulatable**. It is possible to efficiently simulate an accepting transcript without knowledge of w. For Schnorr, this is accomplished by randomly choosing $\hat{d} \in \mathbb{Z}_q$, randomly choosing or taking as input $\hat{c} \in \{0,1\}^t$, and computing $\hat{b} = g^{\hat{d}} y^{-\hat{c}}$. Note that the simulator requires knowledge of d before producing b, and so if the Σ-protocol is executed in the proper order, $\mathcal{V}$ is assured that $\mathcal{P}$ knows w.

- **Special Honest Verifier Zero-Knowledge**. Let $\langle b, c, d \rangle$ be an accepting transcript of an execution of the protocol and $\left\langle \hat{b}, \hat{c}, \hat{d} \right\rangle$ be a simulated accepting transcript. The transcripts are identically distributed. This fact can be used to prove that $\mathcal{V}$ cannot learn anything about w as long as $\mathcal{V}$ honestly chooses random challenges.
- **Public Coin**. Since $\mathcal{V}$ only sends a random value, Σ-protocols are a public-coin protocol.

Logical Conjunction. It is efficient to construct logical conjunctions or disjunctions for statements proven with a Σ-protocol [CS97, Bra97]. A conjunctive version of the Schnorr protocol is due to Chaum and Pedersen [CP92]. Let g_1, g_2 be generators and y_1, y_2 be group elements. The prover claims to know a witness w such that $y_1 = g_1^w$ and $y_2 = g_2^w$. $\mathcal{P}$ selects $a \in_r \mathbb{Z}_q$ and computes $b_1 = g_1^a$ and $b_2 = g_2^a$. $\mathcal{V}$ selects a single challenge $c \in_r \{0,1\}^t$. $\mathcal{P}$ then computes $d = cw + a \pmod{q}$. $\mathcal{V}$ accepts if both $g_1^d = b_1 y_1^c$ and $g_2^d = b_2 y_2^c$.

In the original Schnorr protocol, the witness w must exist given the properties of $\mathbb{G}_q$ and so it is simply a proof of knowledge of w. Here, a w satisfying the relation may not exist. Therefore it is a proof of membership for the relation in addition to being a proof of knowledge. Observe also that $\langle g_1, g_2, y_1, y_2 \rangle$ forms a DDH-tuple. Therefore this proof is equivalently a proof of membership for DDH-tuples. Finally recall also that an Elgamal ciphertext and a rerandomization of it form a DDH-tuple. Therefore this proof can also be used to prove the correct rerandomization of an Elgamal ciphertext.

Logical Disjunction. A disjunctive version of the Schnorr protocol is due to Cramer *et al.* [CDS94]. Once again, let g_1, g_2 be generators and y_1, y_2 be group elements. Now the prover claims to know either witness w_1 or w_2 such that $y_1 = g_1^{w_1}$ and $y_2 = g_2^{w_2}$.

Recall that Σ-protocols are simulatable and that a sound transcript is one that is executed in the correct order (specifically, the verifier assures that the challenge c is provided only after the prover sends b). The trick used for disjunction is to issue two accepting transcripts, one for each witness, but one transcript will be simulated and one will be executed in the correct order. For each transcript, $\mathcal{P}$ may use any challenge but these challenges must add up to (modulo 2^t) the challenge provided by $\mathcal{V}$ after receiving each b. This constrains $\mathcal{P}$ to only being able to choose one challenge ahead of time: the other will be fully determined by $\mathcal{V}$'s choice of c. However $\mathcal{V}$ will be unsure of which is which, so $\mathcal{V}$ learns that $\mathcal{P}$ knows one of the witnesses (at least) but not which.

Without loss of generality, say $\mathcal{P}$ knows w_1. For the second relation involving w_2, which $\mathcal{P}$ does not know, $\mathcal{P}$ prepares a simulated accepting transcript using his own challenge drawn at random: $c_2 \in_r \{0,1\}^t$. Let this transcript be $\langle b_2, c_2, d_2 \rangle$. For the first relation, he follows the protocols: picks a_1 and computes $b_1 = g_1^{a_1}$. He sends $\langle b_1, b_2 \rangle$ to $\mathcal{V}$, and $\mathcal{V}$

responds with $c \in_r \{0,1\}^t$. $\mathcal{P}$ sets $c_1 = c - c_2$ and computes $d_1 = w_1 c_1 + a_1$. $\mathcal{P}$ sends $\langle d_1, d_2 \rangle$ to $\mathcal{V}$, who checks that both transcripts are accepting.

Malleability. Σ-protocols are malleable. In the case of Schnorr, let $\langle b, c, d \rangle$ be an accepting transcript for witness w and common input $\langle q, g, y \rangle$. The following will also be accepting transcripts: $\big\langle (q, g, y), (w), (bg^\beta, c, d + \beta) \big\rangle$ and $\big\langle (q, g, yg^\beta), (w), (b, c, d + \beta c) \big\rangle$. The combination of two accepting transcripts, $\langle b_1 \cdot b_2, c_1 + c_2, d_1 + d_2 \rangle$, is also an accepting transcript. There are many other forms. Given transcripts are already simulatable, this property should not seem surprising or particularly useful, however we will return to it after discussing the Fiat-Shamir heuristic.

Fiat-Shamir heuristic. Recall that Σ-protocols are public coin protocols: $\mathcal{V}$ only provides random bits. The idea of the Fiat-Shamir heuristic [FS86] is to let $\mathcal{P}$ run the Σ-protocol himself, produce an accepting transcript, and let anyone verify its soundness. The issue for someone wanting to verify a non-interactive transcript is not just that it accepts, it is also to determine if it was simulated or generated honestly with $\mathcal{P}$ computing b before choosing c.

The trick is to set $c = \mathcal{H}((q, g, y), b)$, where $\mathcal{H}$ is a cryptographic hash function. If we assume $\mathcal{H}$ is a random oracle, it is intractable to find a b such that $b = g^d y^{-\mathcal{H}((q,g,y),b)}$ (for sufficiently large outputs). Through programming the random oracle, transcripts can still be simulated in an ideal world. The Fiat-Shamir heuristic produces a transcript that is non-interactive, transferrable (anyone will find it convincing), and with care, non-malleable. Note that due to the second example of malleablity above, it is important to include (q, g, y) in the hash query, not just b, if non-malleablity is important [Gen95].

Since non-interactive transcripts will be found convincing by anyone, this opens some possibilities. One is for blinded transcripts, where $\mathcal{P}$ proves a relation to $\mathcal{V}$ in a way that $\mathcal{V}$ can output a non-interactive transcript that proves the relation, however $\mathcal{P}$ never sees the transcript itself and cannot recognize it if he does see it later [LRSW99]. This uses the malleability of the underlying Σ-protocol in a way similar to the examples above.

Beacons. An alternative method for creating a non-interactive proof is to use a publicly known random challenge (that was unknown to $\mathcal{P}$ prior to sending b). While not generally useful in Σ-protocols, beacons are useful in cut-and-choose protocols where the challenge space is small and Fiat-Shamir cannot be used. See Chapter 6 for further discussion.

Non-transferrability. Transferrable, non-interactive proofs are useful for generating a single proof that convinces any interested verifier (universal verification). However at other

times, we want the exact opposite: that a statement only convinces a single verifier and that verifier cannot convince anyone else. For example, if a voter asks a computer to encrypt his vote, the computer can prove it did it right, however the voter should not be able to convince a vote buyer by showing him the proof.

The naive solution is to return to standard interactive Σ-protocols without Fiat-Shamir, however $\mathcal{V}$ can still use Fiat-Shamir instead of drawing a random c which generates a transferrable transcript. This is consistent with Σ-protocols' special honest verifier zero-knowledge property. Instead, an option is to convert the Σ-protocol into a true zero-knowledge proof, which allows dishonest verifiers and provides non-transferability [CDM00]. This can be accomplished by having $\mathcal{V}$ precommit to c before entering the protocol. $\mathcal{V}$ commits to c with a trapdoor commitment (such as the Pedersen commitment above), and $\mathcal{P}$ releases the trapdoor when sending d (to allow extractability).

Designated-Verifier Proofs. Zero-knowledge proofs are interactive and non-transferrable, while Fiat-Shamir is non-interactive and transferrable. A third option is a designated-verifier (DV) proof, which is non-interactive and non-transferrable [JSI96]. DV proofs also employ a trapdoor commitment, but the trapdoor is $\mathcal{V}$'s private key. The $\mathcal{P}$ issues a non-interactive, Fiat-Shamir based proof that either the statement is true or $\mathcal{P}$ knows $\mathcal{V}$'s private key. From $\mathcal{P}$, this proof is convincing since $\mathcal{V}$ will know the latter clause cannot be true. However if $\mathcal{V}$ transfers the proof to an adversary $\mathcal{A}$, either clause could be true and $\mathcal{A}$ cannot distinguish which (the proof is witness-indistinguishable).

Other Σ-Protocols. We have provided examples of knowledge of a discrete logarithm. Other efficient Σ-Protocols exist that we make reference to. One is knowledge of a representation: knowledge of $\langle w, r\rangle$ such that given $\langle g, y, r\rangle$, $y = g^w h^r$ [Oka92]. Another is an interval or range proof: membership of w in the interval $[0, j]$, such that given $\langle g, y\rangle$, $y = g^w$ and $w \in [0, j]$ [Bou00]. Finally, there is also knowledge of an Elgamal plaintext [Kat03]: knowledge of w and r such that given $\langle g, y, c_1, c_2\rangle$, $\langle c_1, c_2\rangle = \mathsf{Enc}(w, r)$. Knowledge of a plaintext can be for regular or exponential Elgamal.

2.3.5 Cryptographic Tests

Σ-protocols allow a $\mathcal{P}$ that knows a witness w to prove the truth of a statement without revealing anything beyond the statement's validity. A cryptographic test allows parties who do not know a witness w to determine the truth of a statement without learning anything beyond the statement's validity.

Plaintext Equality Test. Given two (exponential) Elgamal ciphertexts $\langle a_1, a_2 \rangle$ and $\langle b_1, b_2 \rangle$ and a decryption key m-out-of-n threshold-shared among n trustees, $m+1$ trustees want to determine if the ciphertexts encrypt the same message [JJ00]. The first trustee computes $\langle c_1, c_2 \rangle = \langle a_1/b_1, a_2/b_2 \rangle$. If and only if the original ciphertexts are encryptions of the same message, $\langle c_1, c_2 \rangle$ will be an encryption of 1 (this fact is the same for both regular and exponential Elgamal). The first trustee generates a private random blinding factor β_1 and computes $\langle c'_1, c'_2 \rangle = \left\langle c_1^{\beta_1}, c_2^{\beta_1} \right\rangle$. Due to the homomorphic property of Elgamal, this is a valid encryption of plaintext raised to the power β. If the plaintext is 1, the new plaintext is also 1. If it is not 1, it is mapped to a random number. The trustee then proves knowledge and relation of applying β, and publishes this proof and $\langle c'_1, c'_2 \rangle$. Each subsequent trustee repeats the blinding and proof step, and if one is honest, the result is an encryption of 1 if the messages were equal, or a random number otherwise. The trustees use distributed decryption to learn the result.

Part I

Booth Voting

Chapter 3

Related Work on Booth Voting

3.1 Introductory Remarks

The essential ingredient to secret ballot elections is the voting booth. If an adversary were able to monitor all the activities of a voter, the voter could never cast a secret ballot. In Part II, we will consider internet (and other types of remote) voting where a private voting booth can no longer be assumed—a considerable challenge for maintaing ballot secrecy and coercion-resistance. For now, in Part I of this dissertation, we will concentrate on systems that leverage the physical isolation of the voting booth to achieve secret ballot elections.

The first formalization of the role of the voting booth within cryptographic voting was given by Benaloh and Tuinstra in 1994 [BT94]. Until then, many systems had been proposed that had integrity and a weak sense of ballot secrecy: the cryptographic information made public would not passively reveal how a voter voted. However voters could demonstrate how they voted by revealing, for example, the randomness used to encrypt or commit to their vote. Assuming the existence of the voting booth, Benaloh and Tuinstra proposed the first system where voters cannot prove how they voted: what they called a receipt-free system.[1]

In this chapter, we will provide a literature review of cryptographic voting systems appropriate for booth voting. We begin in Section 3.2 with an overview of the components that make up an election: the participants, typical phases, and types of channels used. We then review the early literature in Section 3.3, which implicitly assumes voters are capable of performing mathematical and cryptographic operations, and thus would need computer assistance. In 2004, Chaum [Cha04] and Neff [Nef04] introduced systems that maintained

[1] Recall in Section 2.2, we call this property coercion-resistance and include a discussion on their disputed equivalence.

the cryptographic properties but voters can cast ballots unaided. We review systems of this type in Section 3.4.

3.2 Election Set-up

3.2.1 Participants

Most cryptographic election protocols involve at least two participants: a set of voters and a set of election trustees responsible for generating the tally. When the voters themselves are the trustees, it is often termed boardroom voting [BF85]. Protocols may also include an election authority responsible for registering voters, maintaing a schedule, or other tasks. Finally, the protocols consist of a set of verifiers who ensure the integrity of different components of the election. Verification may be only available to the voters themselves (voter verification), be open to anyone (universal verification), or some composition of the two.

3.2.2 Phases

Most cryptographic election protocols involve a series of phases. The first is a setup, where keys and public parameters are established. Some systems assume an existing public key infrastructure (PKI) for the talliers and/or voters. The setup could also include commitments to the cryptographic representation of the ballots. Next, some systems assume a registration phase where voters are identified as being eligible to vote and are given some token allowing them to cast a ballot. The voters then enter the voting phase where they construct and cast their ballots. Recently, this is accomplished in an asynchronous single round (vote-and-go) but earlier schemes may involve more than one synchronous round. Once the ballots have been cast, the trustees conduct the tallying phase where they count the votes and produce a final tally. The last phase is post-election auditing, where the verifiers examine the transcript of the election and determine if it should be accepted.

3.2.3 Channels

Cryptographic voting systems make some assumptions about the existence and availability of certain types of channels, which we summarize here.

Untappable Channels. We opened this chapter with the concept of a voting booth, which was used by Benaloh and Tuinstra [BT94] to propose a receipt-free system. The following year, Sako and Kilian modelled the voting booth as a two-way physically untappable channel (and showed that only one-way is needed to achieve receipt-freeness) [SK95].

An untappable channel is a stronger notion than an encrypted or secure channel. Both protect from a computationally bounded, passive eavesdropper. However a secure channel may not protect the secrecy of transmitted information from an adversary that adaptively corrupts one party after the information has been transmitted. Once the adversary learns the session key, he can recover the only set of messages that fit the transcript.

Untappable channels are also stronger than a channel with deniability. In a channel with deniability, an unbounded adversary who obtains a transcript of the conversation after the fact cannot decide whether the transcript is real or simulated. However if the communication is to be meaningful, the sender needs to be assured that the receiver is inferring the correct message from all the possible messages the sender could theoretically be sending. Thus a deniable channel does not prevent an adversary from also inferring the correct interpretation (*e.g.,* through corrupting the receiver, observing the order of the communication, *etc.*).

For more on this distinction, see Canetti and Gennaro's result [CG96] on achieving incoercible multiparty computation with deniable encryption, and the subsequent discussion of its relationship to receipt-freeness by Hirt and Sako [HS00] (and others [CGS97, MN06]).

Broadcast Channels. Many cryptographic voting systems assume the existence of a broadcast channel. More specifically, it is often an append-only, order-preserving (or time-stamped) public broadcast channel called the bulletin board [BF85]. The exact requirements differ between systems but the bulletin board is often used as a place to post messages, commitments to messages, or proofs for universal verification. Constructions for bulletin boards have been provided by Peters [Pet05] and, seemingly independently, Heather and Lundin [HL08].

Anonymous Channels. Some voting systems assume the existence of an anonymous channel. Generally, the anonymous channel will hide the identity of the sender of a message from the receiver, unless otherwise specified. Systems may utilize a specific construction, such as a mix network [Cha81] or a DC-Net [Cha82], both due to Chaum. However using a specific construction may bring additional assumptions. For example, mix networks queue a batch of messages before delivering them (in a different order), while an abstract anonymous channel delivers the messages immediately.

Mix Networks. As mentioned, a specific type of anonymous channel is the mix network [Cha81]. Consider the variant proposed by Park, Itoh, and Kurosawa, which is called a rerandomization or reencryption mix network [PIK93]. It is operated by a set of trustees. Voters post encrypted messages (*e.g.,* with Elgamal) on a bulletin board alongside each other's messages. Once a batch of messages is established, the first trustee takes the set of ciphertexts, individually rerandomizes each, and then privately permutes the order of the messages. Each trustee subsequently repeats this process. After all trustees have participated, the ciphertexts are decrypted (*e.g.,* distributively by the trustees).

A verifiable mix network is one where the trustees prove they correctly rerandomized and shuffled the ciphertexts, without revealing the random factors or permutations. There is a large number of papers on constructing verifiable mix networks. Differences in approaches include: the complexity of the proof; the soundness (high versus overwhelming);[2] the resilience to collusion (most offer unlinkablity with one honest node); and if the proof is universally verifiable (*e.g.,* through beacons with cut-and-choose protocols, Fiat-Shamir with Σ-protocols, or non-interactive proofs/arguments based on bilinear pairings).

For a voting system, mix networks should be universally verifiable and as efficient as possible. We advocate compromising in two regards to achieve higher efficiency. First, we are typically assuming more than one trustee is honest in order to allow threshold (*i.e.,* m-out-of-n where $m < n$) decryption, therefore the mix network can be resilient to all but a small subset of trustees not colluding. Second, the overall confidence in the election result will never be overwhelming, only high, because voter participation in simple checks (*e.g.,* correctly posted ballot) will not be universal (see the discussion in Appendix C). Two verifiable mix networks that are fast, unlinkable with more than one honest trustee, and have high confidence are randomized partial checking (RPC) by Jacobsson, Juels and Rivest *et al.* [JJR02] and "almost entirely correct mixing" (AECM) by Boneh and Golle [BG02].

3.3 Protocols for Ballot Casting

We begin our literature review with cryptographic voting schemes that have been designed primarily or entirely at the protocol level, not at the systems level (although some have been deployed more or less directly: see Section 3.3.7). If we treat these protocols as systems, they would implicitly assume that voters can reliably perform arbitrary mathematical or cryptographic computations. However there is a difference between a voter performing an encryption and a voter using a computational device (*i.e.,* a provided DRE or their own smartcard) to perform the same encryption: in the latter case, the device learns the plaintext and the voter cannot be confident that the device operated correctly. In the

[2]More often, both are asymptotically overwhelming in that $\mathbf{Pr}[\text{Correct}] = 1 - 2^{-k}$. However it is a difference between a k of 10 or 20 and a k of 80 or 160.

next section, we review the approaches to resolving these types of issues for human ballot casting. For now, we can consider the present approaches as providing the cryptographic backbone to deployable E2E systems.

In the previous chapter, we introduced a number of useful and general cryptographic primitives. It is important to note that many of these were developed alongside or after the results surveyed here. For example, many early systems use inefficient cut-and-choose protocols where we today would use a Σ-protocol or zero-knowledge proof. Others rely on heavy communication complexity to distribute the secrecy of ballots, where a modern approach may utilize a threshold encryption scheme. If a protocol seems unnecessarily complicated, it is likely that the direct approach by today's standards was not known.

3.3.1 Anonymous Channels

We begin by reviewing systems that use an anonymous channel.

A first protocol. The first cryptographic protocol was proposed by Chaum as an application of his mix networks [Cha81]. The approach combines (non-verifiable) mix networks with digital signatures. A digital signature allows a signer to sign message m, potentially with randomness r. The signature is computed with the signer's secret key sk: $s = \mathsf{Sign}_{sk}(m, r)$. Using the signer's public key pk, anyone can verify the validity of the signature $\{0, 1\} = \mathsf{Verf}_{pk}(m, s)$.

In Chaum's 1981 protocol, a voter V_i generates a new signature key pair $\langle sk_i, pk_i \rangle$ and submits $\mathsf{Enc}_{pk_t}(pk_i)$ to a mix network under trustees' public key(s) pk_t. The trustees ensure that each submission came from an eligible voter and then process the messages, producing a shuffled list of $pk_{\pi(i)}$, where $\pi(i)$ denotes the composition of each trustee's private permutation. V_i checks that pk_i appears, and if it does, submits vote v_i and $\mathsf{Sign}_{sk_i}(v_i)$ to the mix network. To ensure fairness, the trustees wait until all voters have submitted something to the second round and then mix and decrypt the results. The votes are tallied if $\mathsf{Sign}_{sk_i}(v_i)$ verifies with a pk_i from the first round.

If the voter's pk_i is not correct in the first round, the protocol must be repeated. It should be evident that this round and the signatures can be eliminated by using a verifiable mix network. If the vote is not correct in the second round, the signature will not verify, and if it does not appear at all, the voter can demonstrate a signed vote with a registered key, albeit by losing the secrecy of her ballot. Because of this loss, it is not dispute-free or accountable. It is also not receipt-free as the voter can sell her vote by disclosing sk_i.

DC-Nets. In 1983, Chaum introduced a second type of anonymous channel called a DC-Net. Unlike mix networks, messages are received immediately after they are sent and

provide an easy way to learn the product or sum of all messages. They also require the participation of all senders at every round and secure channels between them. Three notable voting systems used DC-Nets. Chaum introduced a scheme in 1988 that achieved unconditional privacy and correctness based on the RSA assumption [Cha88]. Bos presents a refined version in the DL setting using Pedersen commitments (as "blobs") to commit to a binary (yes/no) vote [Bos92]. As a sketch, votes are encoded as a 0 or 1 and the random factors for the commitment are distributively generated with the DC-Net such that they will cancel out when added together. Voters prove they committed to either a 0 or a 1 with a cut-and-choose protocol with soundness based on the DL-problem. The entire protocol is 5 rounds. Ohta [Oht98] and Boyd [Boy89] also propose schemes similar to Chaum's 1988 scheme.

Reencryption mix networks. Park, Itoh, and Kurosawa, who introduced the reencryption mix network, substitute it for DC-Nets in a scheme similar to Chaum's 1988 scheme [PIK93]. Voter V_i choses their vote v_i and splits it into two shares: *e.g.,* $\alpha_i \cdot \beta_i = v_i$. She submits $\langle \mathsf{Enc}(\alpha_i, r_{i,1}), \mathsf{Enc}(\beta_i, r_{i,2}) \rangle$ to the mix network (where the tuples are shuffled). After mixing, the trustees choose a random ciphertext to decrypt from each tuple. The voter checks that the revealed α_i or β_i is correct. After resolving any disputes, the remaining ciphertexts are opened and a tally is computed from correctly formed pairs. Overwhelming soundness can be achieved by executing through standard parallel composition (*i.e.,* the voter submits and checks k pairs for soundness $1 - 2^k$). While the mixing is verifiable, it is only verifiable to the voters themselves. Pfitzmann outlines some attacks on the exact formulation of the system [Pfi94], most of which can be resolved with current practices[3].

Universally verifiable mix networks. Sako and Kilian present the first universally verifiable mix network in their 1995 scheme [SK95]. It is a cut-and-choose protocol similar to a standard proof of graph isomorphism. To prove list 2 is a shuffled version of list 1, the prover generates a third list. The prover is challenged to show the permutation and random factors between list 1 and list 3, or between list 2 and 3. Soundness can be built by repeating the process with new shadow lists. Sako and Kilian also use this to build a voting system, in conjunction with ideas from the homomorphic secret sharing systems of Benaloh *et al.* We will return to Sako and Kilian's scheme after reviewing these.

3.3.2 Homomorphic Secret Sharing

In parallel to the developments of voting protocols based on anonymous channels, protocols were developed based on a combination of homomorphic encryption and secret sharing.

[3]Using subgroup $\mathbb{G}_q$ and proving knowledge of the plaintext for submissions.

Benaloh schemes. Starting in 1984, Benaloh produced a series of related voting papers [BF85, BY86] culminating in his 1987 dissertation [Ben87]. He introduced the first additive homomorphic encryption scheme, based on the hardness of determining quadratic residuosity in certain composite groups (like RSA groups). The Paillier encryption scheme can be thought of as a generalization, although Paillier allows for arbitrary messages to be decrypted. Benaloh's encryption, like exponential Elgamal, requires a hard problem to be solved to recover the plaintext, which is efficient for small message spaces (such as a tally). He also introduces the idea of a bulletin board.

His schemes are reliant on a simple proof that an encrypted bit $c_b = \mathsf{Enc}(b)$ is a 1 or a 0 (elsewhere he generalizes the proof style as cryptographic capsules [Ben86]). The prover flips a coin and provides either $\langle \mathsf{Enc}(0), \mathsf{Enc}(1) \rangle$ or $\langle \mathsf{Enc}(1), \mathsf{Enc}(0) \rangle$. We will call this tuple $\langle c_1, c_2 \rangle$. The prover is challenged to decrypt $\langle c_1, c_2 \rangle$ and prove their proper form, or show that $\mathsf{Dec}(c_b^{-1} \otimes c_i) = 0$ (*i.e.,* homomorphic subtraction) where i can be either 1 or 2. Voters generate an encryption of their binary (yes/no) vote, prove it is correct, and all ciphertexts can be summed homomorphically.

Techniques for distributed or threshold variants of the Benaloh cipher were not known. Originally, a central authority had the power to decrypt [BF85]. Later, voters would split their vote into n shares that would add up to a 0 or 1 [BY86]. Each of the n trustees would receive a share from each voter and decrypt the sum of these shares. The decrypted sums from each authority, added together, produce the tally. Benaloh then proposes an m-out-of-n threshold variant based on secret sharing [Ben87].

Optimizations. Sako and Kilian in 1994 (not to be confused with their 1995 scheme mentioned above) proposed an optimized version of Benaloh and Yung's distributed protocol [SK94]. Their scheme used partially compatible homomorphisms based on discrete logs, more efficient zero knowledge proofs (although still based on parallelizing proofs with soundness 1/2), and other optimizations. In 1996, Cramer *et al.* further refined this approach using verifiable secret sharing, Pedersen commitments instead of encryption (providing unconditional privacy), and proofs based on Σ-protocols [CFSY96]. Even today, the 1996 Cramer *et al.* scheme is very efficient for the homomorphic secret sharing architecture.

Adding Receipt-freeness. None of these protocols, however, are receipt-free. Receipt-freeness was only introduced in 1994 by Benaloh and Tuinstra [BT94] (the idea was independently developed by Niemi and Renval the same year [NR94]). They proposed two schemes to meet it. The first scheme uses a centralized authority, which gives voters k (*e.g.,* 10) pairs of either $\langle \mathsf{Enc}(0), \mathsf{Enc}(1) \rangle$ or $\langle \mathsf{Enc}(1), \mathsf{Enc}(0) \rangle$. The voter will choose their vote from the first pair; the other $k - 1$ are used to convince the voter that the first pair

is formed correctly. Over an untappable channel, the authority asserts the order for each pair. For each of the $k-1$ pairs, the voter will challenge the authority to either open the challenged pair and show it matches the assertion, or to show that the difference between the corresponding values in the first pair and the challenged pair is zero. These proofs are posted publicly for anyone to check (but not the asserted orders). This provides universal verification that the government is issuing correctly formed ballots (If the voter is an honest verifier and not in collusion with the government), while giving the voter a non-transferrable proof of which ciphertext is which in the first pair. The voter casts the vote of her choice, all casted ballots are added up homomorphically, and the government decrypts the tally.

They also propose a distributed protocol that is similar in structure, only using polynomials with roots 0 or 1 as the votes. The scheme also differs in that voters create their own ballots instead of being issued one. Hirt and Sako later show that a voter can prove how they constructed their ballot to an adversary by exploiting the cut-and-choose protocol used to prove it is correctly formed [HS00].

3.3.3 Mix Networks Revisited

Recall that in 1995, Sako and Kilian introduced the first universally verifiable mix network [SK95]. We now describe their voting scheme. Like in the scheme of Benaloh and Tuinstra [BT94], voters are given a ballot of the form $\langle \mathsf{Enc}(0), \mathsf{Enc}(1) \rangle$ or $\langle \mathsf{Enc}(1), \mathsf{Enc}(0) \rangle$. However in this case, a publicly-known encryption of 0 (*i.e.,* with randomness 0) and of 1 are sent through a reencryption mix network to each voter. Each trustee universally proves they reencrypted the two values and potentially swaps their order, with revealing if they swapped them or not. To the voter, they provide proof of which action they took so the voter can choose the correct ciphertext to submit. Submitted votes are then put back through the mix network, distributively decrypted by the trustees, and a tally is computed. The proof between the trustee and the voter of whether the order was swapped is given over a deniable channel. As discussed earlier, this alone is not strong enough for receipt-freeness but an untappable channel could be substituted. Indeed, Michels and Horster point out one attack exploiting the deniable channel (although for correctness not receipt-freeness) [MH96]. They also find a flaw in an optimized variant of the verifiable mix network where the trustees perform the mixing and the partial decryption in the same step.

As mentioned earlier, many other techniques were developed for verifiable mixing which we do not review. While this literature will often provide voting as an example application, the voting protocols themselves are typically the same: voters encrypt their votes and submit them to the verifiable mix network. Only the details of the proofs change.

3.3.4 Anonymous Channels and Blind Signatures

Blind signatures are used to obtain a signature on a message from a signer, such that the signer never sees the message. A blind signature consists of three additional functions $\langle \mathsf{Blind}, \mathsf{BlindSign}, \mathsf{Unblind} \rangle$ such that $\mathsf{Sign}_{sk}(m, r_1) = \mathsf{Unblind}(\mathsf{BlindSign}_{sk}(\mathsf{Blind}(m, r_2), r_1), r_2)$ and $\mathsf{Sign}_{sk}(m, r_1)$ is indistinguishable from $\mathsf{BlindSign}_{sk}(\mathsf{Blind}(m, r_2), r_1)$. A voter can obtain a blind signature by choosing r_2, called the blinding factor, and is responsible for computing the Blind and $\mathsf{Unblind}$ functions; while the signer chooses r_1 (as appropriate: *e.g.,* in blind signatures based on textbook RSA, r_1 is null) and computes the $\mathsf{BlindSign}$ function.

FOO and follow-ups. In 1992, Fujioka, Okamoto, and Ohta introduced a very well-studied protocol often called FOO [FOO92]. It builds on Chaum's 1981 scheme, where, recall, voters anonymously submit a signing public key in the first round, and their signed vote in the second; both rounds use mix networks. FOO abstracts away from mix networks to general anonymous channels. This adds two complexities: the first is that the election authority can no longer screen unregistered voters from submitting ballots, and second, since anonymous channels are asynchronous, if voters submit their votes over the channel, the receiver sees a running tally before voting is finished, *contra* the fairness property.

FOO addresses these two complexities with a three phase protocol (the first two phases can be held concurrently). In the first registration phase, voters commit to their vote and obtain a blind signature on the commitment from the registrar, who will only sign if the voter is eligible and has not already registered—solving the first complexity. In the second phase, voters submit the commitment and signature over the anonymous channel. This assures that the voting finishes before anyone learns the tally. After voting is closed, the voters submit an opening of the commitment.

Baraani-Dastjerdi, Pieprzyk, and Safavi-Naini address one problem with FOO: if voters do not return for the second phase, the registrar can submit its own votes without detection [BPS95]. The authors solve the problem by issuing a pseudonym to each voter, of which partial information is made public. Horster, Michels, and Petersen distribute the registration authority with blind multisignatures [HMP95], thus require collusion from each trustee to substitute a vote in for an abstaining voter. Ohkubo *et al.* propose using threshold encryption instead of commitments to eliminate the third round, which allows voters to perform the first two phases and "walk away" [OMA+99]. None of these schemes are receipt-free.

Adding receipt-freeness. Okamoto suggests that voters use Pedersen trapdoor commitments in FOO to obtain receipt-freeness [Oka96]. Voters commit to their vote, obtain a blind signature on the commitment, send the commitment and signature over an anonymous channel, and then send the opening over an anonymous and untappable channel to

the authority. The authority will post the opened commitments in a random order and a proof they are correct. Voters can open their commitment to any value. Note that if the authority colludes with the adversary, it can only open the voter's commitment (the voter can prove which commitment is hers by opening it to any value) one way: the way the voter wants. However it is typical to make some assumptions restricting collusion between authorities and adversaries.

A second issue is more subtle: the adversary could supply the voter with a trapdoor commitment to use but not the trapdoor.[4] If the voter were required to send the trapdoor, the authority could change the voter's vote. In a follow-up, Okamoto addresses this issue by distributing the authority into trustees and having the voter anonymously send shares of the trapdoor to the trustees, who can confirm that they interpolate to the trapdoor without recovering it [Oka97]. The paper is also noteworthy for introducing the first formal definition of receipt-freeness. We use this definition in Appendix C. Xia and Schneider also solve this issue by using a paper ballot interface based on Prêt à Voter (see Section 3.4) [XS06].

3.3.5 Homomorphic Threshold Encryption

CGS. In 1997, Cramer, Gennaro, and Schoenmakers introduced an influential voting protocol [CGS97]. They proposed the use of exponential Elgamal, and applied threshold techniques to allow trustees to distributively generate an m-out-of-n shared secret key. Voters encrypt their binary votes as 1 or -1 and use a standard Σ-protocol OR-proof to prove the vote is valid. The encrypted vote is post to a bulletin board and after the election closes, they are added together homomorphically and threshold decrypted. The mechanics of CGS can be seen in paper ballot systems like Prêt à Voter and internet voting systems like Civitas. CGS left two areas to future work: adapting it to multiple candidates and adding receipt-freeness.

Multi-candidate Counters. There are three basic approaches to adapt CGS to more than two candidates. Hirt and Sako propose using a single plaintext [HS00]. If there are v voters, each candidate is given $\lceil log_2(v) \rceil$ bits of the value as a counter. A vote for candidate j from 0 to $L-1$ is encoded as $\lceil log_2(v) \rceil^i$. This counter (HS-counter) increases the message space (and possible tallies) exponentially, making decryption with exponential Elgamal inefficient in general. It is typically used with Paillier. Hirt later proposes using an individual exponental Elgamal ciphertext for each candidate (Hirt-counter) [Hir01]. Peng *et al.* propose using regular, multiplicatively homomorphic Elgamal and assigning each

[4] The voter could instead use a publicly known $\langle g, g^x \rangle$ where it would be clear she did not know x: *e.g.,* Verisign's public key. She could also use $\langle g, \mathcal{H}(g) \rangle$. These may require the values to be encoded into $\mathbb{G}_q$.

candidate a small prime (or co-prime) (PABDL-counter) [PAB+04]. Votes can be multiplied together and as long as the value does not exceed the group size, causing a modular reduction, the result can be easily factored to obtain a tally. Since this quickly limits the number of voters, they propose randomly assigning votes into maximum sized groups and obtaining sub-tallies for these groups. A variant of the PABDL-counter appeared earlier, where the vote accumulator is allowed to exceed the group order and the tally is recovered with an exhaustive search [KY02].

Adding receipt-freeness. In 2000, Hirt and Sako combined ideas [HS00] from Sako and Kilian's 1995 paper (SK) and CGS. Trustees form a mix network and shuffle a list of candidates, proving universally that the shuffle is correct while proving to the voter what the shuffle is using a designated verifier proof over an untappable channel (instead of a deniable channel). Once voters have obtained a ballot, votes are tallied using CGS. They also offer an optimization for binary votes within CGS that is of independent interest for the observation that a binary vote can be homomorphically flipped. This is used later by Neff [Nef04].

In 2000, Lee and Kim suggest an alternative approach [LK00]. Voters encrypt their own vote but have it rerandomized by an independent party called the randomizer. This creates two challenges: who fulfils the randomizer role and how do voters prove their vote is correct if they do not know the randomness (needed to construct a witness). Lee and Kim implement the randomizer as an "honest verifier" that rerandomizes the ballot and proves it did so correctly to the voter. The voter proves the ballot is correct using a Σ-protocol and the randomizer is able to convert this into a proof that the rerandomized ballot is also correct using the maellability of Σ-protocols. However if the two collude, they can simulate a proof for an invalid vote. Magkos, Burmester and Chrissikopoulos (MBC) suggest using a tamper-resistant smartcard as a randomizer [MBC01], with the assumption that the channel between the voter and the device is untappable. As with Lee and Kim's protocol, the randomizer reencrypts the voter's vote and a joint proof of ballot validity is generated.

Baudron *et al.* adapt CGS to Paillier using the HS-counter and structure the election with a local, state, and national level (each level as a key) [BFP+01]. Voters submit the same ballot to each level and prove they are the same, as well as proving the validity of the ballot. They consider extensions to the system and independently propose the idea of a randomizer to solve receipt-freeness. Its implementation is left open. They also suggest using divertable zero-knowledge proofs as a solution to the voter and randomizer sharing a witness to the plaintext.

Hirt extends the 2000 Lee and Kim protocol [Hir01] (republished [Hir10]). Hirt points out that in LK (the observation applies as well to MBC), Σ-protocols are used under the honest-verifier assumption, allowing voters to transfer the proofs (*i.e.,* using Fiat-Shamir).

Like Baudron *et al.*, Hirt uses a divertable zero-knowledge proof which is not honest verifier. He also extends the election to K-out-of-L candidates, introducing the Hirt-counter. Voters encrypt each component, prove each is a 0 or 1, and that they sum to K. The randomizer is an entity that is not trusted for privacy or integrity, assuming no collusion between the randomizer and coercer. In 2002, Lee and Kim combine ideas from MBC and Hirt to produce a new protocol [LK02] suitable for K-out-of-L candidates (using an HS-counter). It uses a trusted hardware randomizer and more efficient divertable proofs.

Lee *et al.* combine the 2002 Lee and Kim protocol with mix-network based tallying [LBD+03]. With mix networks, proofs of ballot validity can be avoided since ballots are individually decrypted after mixing (however voters may be able to tag their ballots [HS00]). Voters encrypt their votes, use a randomizer, and receive a designated verifier proof from the randomizer. They do not address independence (voters can copy other voters' ballots). Aditya *et al.* extend the protocol with a different verifiable mix-network, a distributed election authority, and batch verification for some repetitive proofs [ALBD04].

Optimizing CGS with Paillier. Fouque, Poupard and Stern develop threshold techniques for Paillier and apply them to CGS as an example application [FPS00]. As described previously, Baudron *et al.* also use Paillier for implementing an HS-counter. Damgard and Jurik [DJ01] generalize Paillier and develop threshold techniques for their more efficient Paillier variant. They suggest a CGS variant as an example application with tweaked proofs for ballot validity, and suggest making it 1-out-of-L with an HS-counter. The following year, they offer further optimizations [DJ02]. The modified protocol reduces the ballot size, shifts some of the computational burden from the voters to the tailliers, and considers issues with voting from an untrusted computer. Damgard, Groth, and Salmonsen [DGS03] and Groth and Salmonsen [GS04] further extend these ideas, improving the efficiency of the proofs and introducing a secondary channel (*e.g.,* postal mail) to protect against a malicious computer. Like CGS, these variants do not consider receipt-freeness.

Optimizing CGS with Pairings. Boneh, Goh, and Nissim propose a pairing-based encryption scheme, commonly called BGN [BGN05]. Like exponential Elgamal, it is additively homomorphic and requires computing a discrete logarithm to decrypt. However it can use a pairing operation to achieve a single homomorphic multiplication. The authors use CGS as an example application, and show that testing if a ballot v encrypts a 0 or 1 can be accomplished non-interactively by homomorphically computing $v' = v(v-1)$ and having the trustees decrypt v' to 0. For batch verification, many v' values can be added together with random coefficients before decrypting. If the value is non-zero, the invalid ballots are eliminated through a binary search.

3.3.6 Miscellaneous

Small-Scale Elections. There are a number of voting schemes designed for small-scale elections, typically where the voters themselves tally the ballots. We consider these board-room voting schemes out of scope [DLM82, Yao82, HT88, PW92, FO98, Sch99, KY02, MMP02, Gro04a, BT08, OI10]. These schemes tend to use general techniques, like general multiparty computations, have high communication complexity, and incur efficiency costs if voters halt prematurely. Some of them can be adapted to large-scale elections with authorities [Sch99, KY02, Gro04a, BT08] but their relative advantage is with small sets of voters running a complete election at once. Some offer the advantage of information theoretic security.

Hybrid Systems. Kiayias and Yung suggest a hybrid election scheme that uses homomorphic tallying for candidates that are listed on the ballot, and a mix network for write-in candidates [KY04, KY10]. Voters submit a tuple of ciphertexts $\langle c_1, c_2, c_3 \rangle = \langle \mathsf{Enc}(m_1), \mathsf{Enc}(m_2), \mathsf{Enc}(m_3) \rangle$ where m_1 is a listed candidate, m_2 is a binary toggle that is 0 for a listed candidate and 1 for write-in, and m_3 is the write-in candidate. Voters prove $((m_1 \text{ is valid} \wedge m_2 = 0 \wedge m_3 = 0) \vee (m_1 = 0 \wedge m_2 = 1)$ using standard Σ-protocols. All the c_1 values from each voter are added homomorphically and decrypted. The c_2 values for a batch of voters can be added together and decrypted: if it is 0 (assuming a low number of write-ins is reasonable), the corresponding c_2 and c_3 values are discarded. Otherwise, the c_3 values in the batch are carried forward to a verifiable mix network.

ANDOS. Nurmi, Salomaa, and Santean [NSS91] and Niemi and Renvall [NR94] propose voting schemes based on all-or-nothing disclosure of secrets (a generalization of oblivious bit transfer). We discuss oblivious transfer later in Chapter 8. These schemes are very general but also inefficient and do not provide meet more advanced requirements like fairness, universal verification, and receipt-freeness.

Ring Signatures. Let $\mathcal{V}_1, \mathcal{V}_2, \ldots$ be the set of registered voters, all with known public keys $pk_1, pk_2, \ldots$. A ring signature allows a member of the set, $\mathcal{V}_i$, to sign a message such that a verifier can only see it was signed by a member of the set; without revealing which member. This does not require the participation, or even knowledge, of the other voters, only access to their public keys. Ring signatures could be used to, for example, replace the signing authority in FOO-type [FOO92] protocols. The challenge in using ring signatures is preventing voters from double voting. Work by Chow *et al.* [CLW07] uses (escrowed) linkable ring signatures, where messages signed by the same member can be linked together but not to the member, to build a cryptographic voting system with integrity and ballot secrecy. Coercion-resistance is achieved with designated verifier proofs.

Scoring Protocols. A number of contributions are designed specifically for scoring protocols other than plurality voting: *e.g.,* preferential [ABDV03], instant-runoff voting [KK08, WB09b], and single transferable vote [TRN08, BMN+09].

3.3.7 Deployment

A few of the protocols we have reviewed have been deployed to run verifiable, but coercible (due to the absence of an untappable channel), internet elections. Examples of these include the following.

- **Sensus** [CC97]: A 1996 system based on FOO [FOO92] without the anonymous channel. Voter cryptography is done client-side with a Javascript applet. While not deployed, it appears to be the first full implementation of a cryptographic voting scheme.
- A 1996 system [DNW96] by Davenport *et al.*, also based on FOO used for student government elections at Princeton. Voter cryptography is done server-side with Perl.
- **E-vox** [Her97]: A 1997 system, also based on FOO, used for undergraduate government elections at MIT. Voter cryptography is done client-side with a Javascript applet and the anonymous channel is implemented as a single mix server. Unlike Sensus, authentication does not require a PKI, just a username and password.
- **InternetStem** [Sch99]: A 1999 system based on CGS [CGS97] (modified for multiple candidates) used for a shadow election of the Netherlands national election. Voter encryption and Σ-protocols are done client-side with a Javascript applet.
- **Votopia** [Kim02]: A system based on Ohkubo *et al.* [OMA+99]. Used in a viewer poll to select the most valuable players in the 2002 World Cup. Voter cryptography is done client-side with a Javascript applet.
- A 2002 system [FMM+02, FMS10] designed by NEC researchers, roughly based on Sako and Kilian's 1995 scheme [SK95] with more efficient proofs. Used frequently since 2004 in a 20,000 person organization. Voter cryptography is done client-side with a Javascript applet.
- **Helios** [Adi08]: A 2008 system with implementations varying between mix networks and homomorphic encryption. Used in the Université catholique de Louvain and Princeton student elections in 2009 [AMPQ09]. Voter encryption and Σ-protocols are done client-side with Javascript.

These systems are suitable for low-coercion scenarios where voters trust the computer they are voting from for both privacy and correctness (with the exception of Helios, which provides a mechanism to challenge the correctness of the machine). In Part II, Chapter 9 we will examine systems for use in high-coercion scenarios in detail, and also review systems addressing the untrustworthiness of a voter's personal computer.

3.4 Protocols for Unaided Ballot Casting

The previous section outlines many protocols that implicitly assume the voter is capable of reliable computation, while the receipt-free protocols additionally assume an untappable channel. In this section, we examine work done on ballot casting protocols for computationally unaided voters. We will only consider schemes with receipt-freeness (or coercion-resistance). The voter may use a reliable computer prior to ballot casting to construct some non-secret information to bring into the polling station, and she is nearly always required to verify certain proofs after her ballot is cast, but for the window of time in the voting booth, she is unassisted.

More generally, the voter may be computationally assisted during a pre-casting and a post-casting phase but these phases are tappable. Between these phases, the voter has temporary access to an untappable channel but cannot be computationally assisted by her own trusted device. Within this window of time, the voter must construct and submit her vote and be convinced that it is correctly formed. We will consider systems of two types: DRE systems where voters interact with a computer terminal that the voter does not trust and paper-based systems where voters mark a paper ballot, which could be subsequently scanned by a scanner the voter does not trust.

3.4.1 A Framework

We present a general framework for human ballot casting protocols. While it does not capture all variations, it should assist in communicating the basic idea and we will refer to it occasionally throughout the dissertation. The election verification can be broken into three steps.

(a) **Marked-as-Intended:** The voter produces and retains a verifiably correct obfuscation of their vote, such that given only the obfuscated vote, it is not possible to determine the vote.
(b) **Collected-as-Marked:** Obfuscated votes are collected by the election authority, published publicly, and voters check that the obfuscation of their vote is included and correct in this collection.
(c) **Counted-as-Collected:** Obfuscated votes are collectively deobfuscated to produce a tally in a way that is verifiably correct and does not reveal the link between any obfuscated and deobfuscated votes.

The composition of these properties results in the voter's preference being *counted-as-intended* (*i.e.,* verified end-to-end). In parallel to each of these three steps, the secrecy of

the voting preference should hold. The challenge for E2E human ballot casting protocols is performing (a) unaided. The literature proposes at least five approaches to obfuscating a vote:

1. Having the vote encrypted and retaining a proof of correctness,
2. Applying a permutation to the vote and retaining partial information,
3. Substituting a code for the vote and retaining the code,
4. Splitting the vote into shares and retaining a share, and
5. Swapping the vote for another voter's (unobfuscated) vote and retaining it.

Generally, DRE-based E2E systems use 1 or 3, while paper-based E2E systems use 2–5. The difference between 1 and the protocols from the previous section is that the voter cannot trust that her vote was encrypted correctly. Further, while she can retain evidence that she can use later, when computationally aided, to verify it was properly encrypted, this evidence cannot constitute a transferrable proof of how her vote was encrypted.

Terminology note. Somewhat confusingly, the obfuscated vote retained by the voter in step (a) of the framework is often called a "receipt." This contradicts the earlier notion of "receipt-freeness," where a receipt was a transferrable proof of the vote. Here a receipt could preserve the secrecy of the ballot. To be consistent with the literature, we will henceforth use the term "receipt" as the privacy-preserving receipt retained by the voter, and refer to "receipt-freeness" as incoercibility or coercion-resistance.

3.4.2 DRE-based E2E Systems

Votegrity. The first unaided, human-verifiable E2E system was Votegrity, developed by Chaum [Cha02, Cha04]. The scheme uses visual cryptography (we omit details of visual cryptography here, however it is covered later in Section 8.2). After the voter casts her vote on the DRE, two receipts are printed on transparencies. When the voter overlays the transparencies, her vote is displayed. However each sheet individually does not reveal any information about her vote. She is free to retain either transparency as a privacy-preserving receipt and destroys the other. Each transparency also contains sufficient information to generate the other layer, but this information is cryptographically protected. Votegrity provides the basis for a number of paper-based E2E systems, and in later examples, we will provide more details on how receipt systems like this one interact with the underlying cryptographic protocol.

MarkPledge and variants. Neff's MarkPledge scheme uses cryptography more directly [Nef04]. The voter uses a trusted device to commit to a challenge that will be used in an interactive proof. She provides the commitment and specifies her choice of candidates. For each candidate, the DRE will print a row of ciphtertext pairs. The length of the row is proportional to the soundness of the protocol. Each ciphertext encrypts either a 0 or a 1, and each ciphertext pair contains matched encryptions for the selected candidate and unmatched encryptions for the others.

For example, if the voter votes for Bob in a race between Alice, Bob, and Carol, then the ciphertext pair (assume a row of length 1 for simplicity) beside each respective candidate might be $\langle \mathsf{Enc}(0), \mathsf{Enc}(1) \rangle$, $\langle \mathsf{Enc}(1), \mathsf{Enc}(1) \rangle$, and $\langle \mathsf{Enc}(1), \mathsf{Enc}(0) \rangle$. The DRE then pledges to the matching bits it used for Bob's ciphertext pair: 1. This "pledge" is provided to the voter over the untappable channel but is not printed. The voter then challenges the DRE to open either the left or right ciphertext within each pair. For example, for the right, the DRE obliges and reveals $\langle \mathsf{Enc}(0), 1 \rangle$, $\langle \mathsf{Enc}(1), 1 \rangle$, and $\langle \mathsf{Enc}(1), 0 \rangle$. The voter verifies that for the candidate she voted for, Bob, the revealed ciphertext matches the pledge (the DRE can always pledge correctly if it correctly encrypts matched bits for the selected candidate). Note this does not yet prove that the other candidates *do not* have matched ciphertexts (*i.e.,* the ballot is not overvoted).

With a larger row of length ℓ and independent challenges (right or left) for each, the soundness of the selected candidate having matched ciphertexts is $1 - 1/2^{\ell}$. Neff then uses an exponential Elgamal trick, also used by Hirt and Sako [HS00], to homomorphically flip, for each pair, the remaining ciphertext if the revealed plaintext is a 0. This results in $\mathsf{Enc}(0)$, $\mathsf{Enc}(1)$, and $\mathsf{Enc}(0)$ for our example, and one can verify that it will always lead to $\mathsf{Enc}(0)$ for unmatched/unselected pairs and $\mathsf{Enc}(1)$ for matched/selected pairs. A simple disjunctive Σ-protocol can prove there is only 1, finally establishing that ballot was not overvoted. We are left with a $\mathsf{Hirt\text{-}counter}$ which can be tallied (we actually have ℓ counters but we can generate ℓ identical tallies, one for each column, in parallel as an additional check).

Slight variations on the MarkPledge protocol are presented by Karlof, Sastry, and Wagner (who point out some system-level attacks and covert-channels in both MarkPledge and Votegrity) [KSW05] and Adida and Neff [AN06]. Adida and Neff later present MarkPledge2, which offers a more compact representation of each vote that eliminates the randomness in MarkPledge that could be used as a covert channel [AN09].

Moran and Naor present a variation of MarkPledge where the voter specifies the candidate and the DRE prints a (perfectly hiding) commitment to the candidate [MN06]. It then uses a Σ-protocol to prove the commitment matches the selected candidate while simulating proofs for the other candidates. The voter provides her challenges ahead of time for all the candidates except for the selected one. The DRE does the first move for each but prints the commitments on a receipt tape that is physically hidden behind a screen

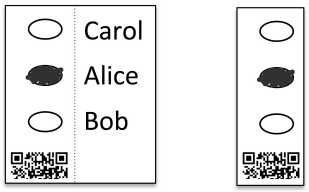
(as the tape advances, the printing becomes visible). This prevents the voter from employing the Fiat-Shamir (or a related) heuristic to make the transcript transferable. The voter then provides the challenge for her selected candidate, and the DRE completes each proof. At the bottom, it records the challenges it used for each candidate in the order of the candidates. The voter verifies that the challenge for her selected candidate is recorded correctly, and later, aided by a computer, will verify that the transcript for each candidate is accepting.

Voter-Initiated Auditing. Benaloh proposes a simpler cut-and-choose protocol to obtaining an encryption of a vote [Ben06, Ben07]. In this scheme, the voter specifies her candidate, and the DRE encrypts the selection and prints it on a paper card (which remains inserted in the machine or is otherwise not visible). The voter can then chose to audit this card, in which case the randomness used to encrypt the selection is printed, or to keep it, in which case it is signed and can be submitted. Independently, Gardner, and Rubin [GGR09] and Feldman and Benaloh [FB09] propose variants to eliminate the covert channels in the randomness of the encryptions: the former using a verifiable random function and the latter using precommitted ciphertexts. Voter-initiated auditing is used in the VoteBox DRE system [SDW08] and the Helios internet voting system (where the voter, or any auditor, is challenging her webbrowser) [Adi08].

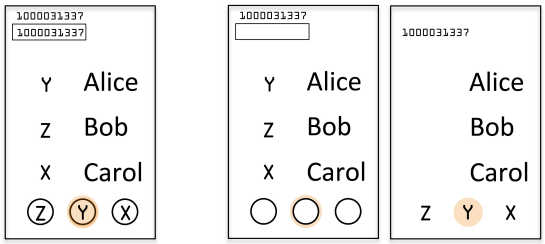
Bingo Voting. Bohli, Müller-Quade, and Röhrich present Bingo voting, a DRE that depends only on a trusted true random number generator (TRNG) [BMR07]. Voters are presented with a list of candidates and a random-looking, but precommitted to, code appears beside each. The TRNG generates a random code of the same length. The voter makes their selection and the DRE replaces the pre-committed code beside the selected candidate with the one generated by the TRNG. The voter retains a copy of the candidates and codes. During tallying, all pre-committed codes that do not appear on a receipt are opened: since a vote for a candidate causes that code to be unused, the number of opened commitments for a candidate will be the number of votes received (plus the number of abstaining voters). It additionally uses a cut-and-choose protocol to show that each remaining unopened commitment appears exactly once on some receipt in the election.

3.4.3 Paper-based E2E Systems

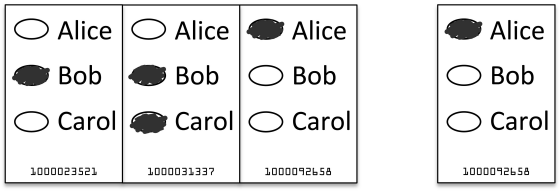
Prêt à Voter. The first unaided paper-based E2E system was Prêt à Voter, proposed by Ryan [Rya04, CRS05]. The voter is presented with a paper ballot, where the order of the candidates has been independently randomized. The voter marks her preference and splits the ballot into halves, where one half shows the position she marked and the



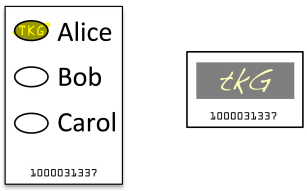
(a) Prêt à Voter



(b) Punchscan



(c) ThreeBallot



(d) Scantegrity

Figure 3.1: Examples of obfuscation techniques for E2E paper ballots: permutations in (a) Prêt à Voter and (b) Punchscan, splitting in (c) ThreeBallot, and code substitution in (d) Scantegrity. For each, a marked ballot and receipt is shown.

other shows the candidate ordering (see Figure 3.1(a)). She retains the first half as her receipt and shreds the second. The receipt contains cryptographic information necessary

to reconstruct the permutation.

If the receipt contains an encryption of the permutation, it can be paired with the mark position, and this tuple can be put through a mix network together. The main challenge with Prêt à Voter is how the permutation is distributively generated and if the mark position can be combined with the encrypted permutation before mixing. For example, Ryan and Schneider suggest that if the permutations are limited to cyclic shifts, the shifts could be an encryption of a random number (taken modular the number of candidates, mod $|C|$, after decryption), and the mark position can be added in homomorphically as an offset [RS06]. With exponential Elgamal, the challenge is keeping the randomly distributed numbers within an interval that can be decrypted. Xia, Schneider, and Heather suggest the use of Paillier to avoid this [XSH08]. Kusters, Truderung, and Vogt show a simplification for distributively generating the cyclic shifts [KTV09].

Using cyclic shifts (1 out of $|C|$) can lead to attacks. For example, if the canonical order of candidates is Bush, Buchanan, and Gore, and the main contenders are Bush and Gore, adding an extra cyclic shift to random ballots[5] will push many Bush votes onto Buchanan, a few Buchanan votes onto Gore, and many Gore votes onto Bush. Between Bush and Gore, the attack is relatively beneficial to Bush. Using full permutations (1 out of $|C|!$) can be inefficient to generate. Ryan and Teague show how to generate orderings based on Latin squares (1 out of $|C|^2$) that avoids this attack [RT09a].

An additional concern with Prêt à Voter ballots (and the ballots of other E2E paper-based systems) is that the printed permutation matches the encrypted permutation. Voters can request a ballot to be print audited for correctness, however the randomness used to encrypt the permutation must be available. Adida and Rivest propose Scratch & Vote, a Prêt à Voter variant that puts the opening of the encrypted permutation under a scratch-off surface [AR06]. If the ballot is cast, this information is destroyed. Note that we are also implicitly assuming a trusted printer that learns the permutation when printing the ballots (a problem we preliminarily address in Chapter 8).

Permuting the list of candidates is of independent interest. It eliminates the impact of a known cognitive bias called the primacy effect, where candidates that appear higher in the list are statistically significantly more likely to receive votes (with the first listed candidate receiving the highest benefit) [KS04]. On the other hand, many election districts require ballots to be printed in a specific order making randomized orderings illegal (the same applies to serial numbers and other unique information in many districts as well).

Punchscan. The Punchscan system invented by Chaum is also based on a permutation, however it is spread across two sheets of paper [PH06, FCS06, ECCP07b, CCEP07]. The

[5]The system provides independent detection mechanisms for this attack, but the attack does target what is generally the weakest link in systems of this kind: the receipt-check (see also Section C.1.2).

candidates are listed in canonical order and, additionally, a column of randomly ordered symbols are printed on the top layer beside the candidates' names and a row of matching but randomly ordered symbols is printed on the bottom layer underneath the candidates (see Figure 3.1(b)). These bottom layer symbols are visible through circular holes in the top layer. A voter marks a ballot by finding the symbol in the bottom row that corresponds to the symbol beside their preferred candidate and daubs this position with a suitably-sized Bingo dauber such that the ink is clearly visible on both layers of the paper. The voter keeps one of the sheets as a receipt and shreds the other. An advantage of the two layer ballot is when a voter checks her receipt, she is simultaneously print auditing half of the ballot. This eliminates the need for the separate print audit in Prêt à Voter and Scratch & Vote (and Scantegrity below). The underlying cryptography of Punchscan is based only on commitments and is very similar (in fact, interchangable) with the Scantegrity system we will present in Chapter 4. Our optimization of this verification system, Eperio, is also compatible with Punchscan and is presented in Chapter 7.

Scantegrity. The Scantegrity system, propsosed by Chaum *et al.* uses code substitutions to obfuscate the vote. Voters are issued a paper ballot and mark the position beside their selected candidate using a special pen (see Figure 3.1(d)). The mark position contains a confirmation code in invisible ink and the pen develops the ink so that the code is visible. Further details follow in Chapter 4.

ThreeBallot. The ThreeBallot system, invented by Rivest, uses a physical analogue to homomorphic secret sharing [RS07]. Voters are given a multiballot that consists of three ballots which, except for a varying serial number, are identical (see Figure 3.1(c)). The voter assigns a single vote to each candidate and a second vote to the candidate she selects. She then retains a copy of one of the ballots as a receipt. Since *on at least one receipt*, each candidate she did not vote for has a mark and the candidate she did vote for is missing a mark, the marks on the receipt do not provide complete information as to how she voted. The ballots are split up and posted on a bulletin board. The voter can check that her ballot appears unmodified, and all the ballots can be summed to produce a tally (after subtracting the number of votes from each candidate's total).

ThreeBallot relies on a trusted ballot validation device to make sure ballots are formed correctly (otherwise it is possible to cast double or negative votes). In most E2E systems, an adversary gains no advantage from examining a voter's receipt. In ThreeBallot, this is not the case. Clark, Essex, and Adams point out that the integrity of the board relies on an adversary not knowing which ballots were retained as a receipt (if the adversary modifies a ballot that was not retained, it will not be detected) [CEA07]. The more receipts the adversary sees, the greater the probability he can evade detection. We also point out that ThreeBallot receipts can leak partial information about how the voter voted, within

a natural interpretation of how a voter marks their ballot. However note that switching the order can eliminate this bias [MPQ07] (the order of actions is often important in preserving secrecy: see Section 5.4.2 for another example). Others have pointed out that a given receipt cannot be legally combined with any arbitrary pair of ballots on the board (especially in multiple contests). This can lead to a reconstruction attack [HSS09] where a receipt can only belong to a small or singleton set of other ballots.

Aperio. The Aperio system, proposed by Essex, Clark, and Adams, also uses a permuted candidate list [ECA08, ECA10]. It is a paper-based system designed for developing countries without any electronic infrastructure. The ballots are constructed from four-layered carbon paper. The top layer is a normal paper ballot, the second layer records only the mark position (copied through from the top layer) and not the candidate list. This layer is kept as a receipt (it is essentially a Prêt à Voter receipt). The bottom two layers contain the mark position plus special serial numbers that are used in the post-election audit. We present an electronic version of Aperio, Eperio, in Chapter 7.

Farnel and Twin. Another obfuscation technique is exhibited in Farnel [Cus01] and Twin [RS07]. In both of these systems, the voter casts a normal paper ballot with a unique serial number. Before she casts her ballot, a copy of it is made and added to a basket[6] or hopper. The voter can then obtain a copy of a random voter's ballot from the basket as a receipt (with small probability, it will be her own but she cannot prove it is hers). After the election, all the ballots are posted publicly and the distributed receipts can be used to check the public list. In this basic variant, chain-of-custody is still required over the basket. If the basket were wholesale exchanged for modified ballots, the board could present the modified tally without detection. Of course, if chain-of-custody can be assumed, there is no need for the verification mechanism. Farnel has been extended with cryptographic techniques to provide true E2E verification [ACG06, AR08, ACG10].

[6]Farnel means basket in Portuguese

Chapter 4

Deploying Punchscan and Scantegrity

This chapter references and excerpts published work co-authored with the Punchscan team and the Scantegrity team on the Punchscan case study [ECCP07a], Scantegrity election system [CCC+08, CCC+09], and Scantegrity case study [CCC+10] (with minor reference to co-authored work on Scantegrity predecessors [PCE+10, CEC+08] and a mock election that preceded the case study [SCC+10]).

4.1 Introductory Remarks

In this chapter, we trace through an iteration of deploying, redesigning, and redeploying an E2E system. We begin by describing an election we ran using the Punchcan system. We then show how lessons and experiences from this election contributed to the design of the Scantegrity system. Scantegrity was selected by the municipality of Takoma Park for use in their 2009 municipal election. We document the modifications we made to the system to prepare for the election, how feedback from a mock election further refined the system, and finally the results of the actual election.

This chapter shortcuts a number of important steps in the development of both systems. For example, what we actually describe as Scantegrity is technically the Scantegrity II system, which added the use of invisible ink to an earlier design. Also while we review the most significant elections ran with both systems, we also did run a number of small-scale, non-binding elections with both systems.[1]

Contributions. The contributions of this chapter can be summarized as:

[1] These are documented on `punchscan.org` and `scantegrity.org`.

- A case study of the first binding election to use a paper-based E2E system (with Punchscan),
- Scantegrity: a new E2E add-on system that can be interfaced with common optical scan systems,
- The addition of strong dispute resolution procedures to Scantegrity, and
- A case study of the first governmental election to use a paper-based E2E system (with Scantegrity).

4.2 Case Study: Punchscan

In 2007, we employed Punchscan (see Section 3.4.3) in its first binding election. The election was run by the Graduate Students' Association / Association des étudiant(e)s diplômé(e)s (GSAÉD)[2] at the University of Ottawa to elect candidates to positions within GSAÉD, as well as conduct a referendum on one bylaw. Among the reasons GSAÉD cited for their decision to use Puncshscan was the desire to speed up the tally process, increase the integrity of election results, provide a means to identify double-voting, and, at an academic level, play a leading role in voting systems research.

4.2.1 Requirements

The election consisted of five polling stations. Four were located on the main University of Ottawa campus, while one was located on a satellite campus (within Ottawa) for the medical students. The election was held over two days. There were approximately 1000 eligible voters and turnout was only 154.

Ballot. Being a fully bilingual university, the ballot was required to be worded in both English and French. At our recommendation, a bilingual ballot was used instead of different sets of ballots for each language (although our system could accommodate both). The ballot consisted of six contests. Five were positions for office and one was a referendum. Of the five office positions, only one was contested. However the uncontested officials still needed to be confirmed by a majority of voters according to the GSAÉD regulations. Thus these four contests consisted of a 'yes' or 'no' option. The contested office position had two candidates, and the referendum also consisted of a 'yes' or 'no' option. In essence, the election consisted of six contests, each of which had two candidates/options.

[2]This author was a member of GSAÉD at the time.

Double voting. In past elections, voters were not assigned a particular polling place. Each polling place had a list of eligible voters and maintained a local copy of which voters had been issued a ballot, however there was no way to prevent a voter from voting at each polling place. Such "double voting" could be detected later when the lists were combined but the ballot could not be revoked. GSAÉD had experienced double-voting in previous elections and wanted a system to prevent it. Since we were providing laptops at each polling place to operate the scanners, we implemented a basic electronic pollbook that could keep realtime information about who had voted. It was web-based, hosted on Punchscan's server, password-protected and accessible over SSL. Voters were authenticated with their student card. Querying the pollbook with a voter's student number would return one of three possibilities: a message indicating that the student number was not found in the database, an option to mark that individual as having voted, or an alert that that student had voted already.

Chain voting. Although it was agreed to be unlikely to occur in this election environment, the Punchscan team decided to test a countermeasure to the coercion technique known as "chain voting." If an adversary is able to obtain an uncast ballot, he could mark it for his preferred candidate, and coerce a voter by requiring her to return him an uncast ballot. Unless she can herself steal a ballot, she must cast the adversary's ballot as her own and return with the ballot she was issued. This requires her to swap the ballots in the privacy of the voting booth. To prevent this swapping, we developed special clipboards that employ a cylinder lock affixed to the top right corner. Before the voter is presented with an unmarked ballot, that ballot is locked to the clipboard. When the voter returns to cast their vote, the poll worker checks to ensure that the ballot is still locked to the clipboard and that the paper corner is not torn.

Recoverability. In Punchscan, there is no paper record of the ballots. Half of the paper ballot is shredded while the other half is kept by the voter as a receipt. Punchscan only retains a digital record of the receipt by optically scanning the ballot half. GSAÉD had procedures in place for handling paper ballots and also expressed concern over losing the ballots in the event of a power outage, hard drive failure, or other unpredictable event. We agreed to provide a paper record of the same information that is scanned. While this record cannot be hand-counted (it preserves the secrecy of the ballot), it could be used with the underlying cryptographic information to reconstruct the tally.

Election authority. While Punchscan offers support for a distributed election authority, GSAÉD opted to have the chief returning officer (CRO) act as the sole trustee. Punchscan employs a trusted computer to generate the election data (it is trusted only for secrecy not

for integrity). The trustee enters a passphrase and all the election data is deterministically generated from this passphrase. No state is maintained between election events; it is simply regenerated each time it is needed from the passphrase.

4.2.2 The Election

The layout of the Punchscan ballot can be designed in most standard desktop publishing applications. Since each ballot will contain unique information, we use coloured disks or rings to denote where the random symbols should be placed. The layout, in PDF form, is loaded into the Punchscan ballot authoring software, which determines the coordinates of all of the positional markers. It produces an XML drill file, which was used by a machinist to accurately drill the serial number, mark and lock holes in reams of 500 sheets of paper at a time using a computer-positioned vertical mill.

Meetings 1 and 2. The first meeting was held prior to printing the ballots. The chief returning officer entered a description of the election and her passphrase, and the open source software generated the random permutations for each of 3000 ballots and the information needed to reconstitute the vote after one half of the ballot is destroyed. This information is committed to and the commitments are output onto a USB key. This information was then posted publicly on the Punchscan server. The rest of the intermediate values and internal state is purged upon powering down the system. The second meeting was held the following day. Half of the committed information (1500 ballots) was decommitted in a cut-and-choose proof that the election data is correctly formed. To provide universal verifiability, the challenge was generated from stock market data (see Chapter 6). The audit is based on an open specification and any interested party could implement her own auditor.

Ballot printing. The remaining 1500 ballot representations were printed onto paper. The CRO entered her passphrase to (re)generate the information for each ballot and the system output a private XML file containing this information (a shortcoming of all paper-based E2E election systems is the trusted printer problem; we take some initial steps to addressing this in Chapter 8). The ballot authoring software reads this data and overlays it on the ballot template to output the ballots in PDF form. Under the supervision of the trustee, we printed batches of the ballots in parallel on five inkjet printers. The printing was done in colour to allow for coloured scanner registration marks, and the alignment between the top and bottom layers had to be sufficient for allowing the ballots halves to be overlaid. We experienced difficulty with the top sheets jamming due to the holes. Printing took about an hour and upon completion, the ballots were placed into boxes, sealed, and signed along the seal by the CRO.

Ballot issuing. On election day, voters identified themselves and if eligible to vote, were issued a ballot. The poll worker would place the top page of the ballot facedown onto the clipboard assembly followed by the bottom page. The clipboard had a "bookcover" privacy shield that was closed over the front of the ballot and then the assembly could be turned face-up. The privacy shield only showed the two serial numbers (on each sheet), which were checked to match. Finally the ballot was locked onto the clipboard and given to the voter.

Ballot marking. Voters would fill out the ballot in a private voting booth. To mark a contest, the voter located the chosen candidate/option, noted the symbol beside it, located the hole containing the same symbol, and marked the hole by firmly daubing it. Upon returning from the booth, the voter would be instructed to choose either the top or bottom sheet for shredding. This sheet was ripped out of the locked assembly and shredded in view of the poll workers. The assembly with the remaining sheet still locked in was returned to the poll worker.

Ballot casting. The poll worker unlocked the clipboard and removed the sheet. The sheet was scanned with an optical scanner and the software's interpretation of the marks were displayed on a computer screen. The voter approved the marks and cast the ballot electronically. The sheet was then placed in the printer, which printed symbols on the sheet according to the scanning software's interpretation of the marks (*e.g.,* overlaying an X on unmarked positions and an O on marked positions), printed a digital signature of the marks, and printed an additional record of the marks in the corner of the ballot. The corner was cut off by the poll worker and retained as a paper backup (although, like the scanned information, it is not alone sufficient for determining how a voter voted), while the sheet itself was given to the voter as a receipt (recall that any single sheet of the two-layer ballot does not reveal how the voter voted).

Tallying. After the polling stations were closed, the Punchscan team, the CRO, and a scrutineer assembled to generate the results. The ballot receipts were uploaded to the Punchscan server and posted. Through a web-interface, voters can check that their receipts match the posted information. The CRO then entered her passphrase to deterministically regenerate the election data that was committed to prior to the election. Once generated, the tally was computed using this information and the receipt information. The tally and a commitment to its proper computation were also posted on the Punchscan server. As with the pre-election audit, the following day, the tallying procedure was audited using stock market data and a cut-and-choose protocol. This audit is also based on an open specification and is software independent.

4.2.3 Lessons Learned

Over the course of the election, the poll workers experienced some technical failures: the polling place software becoming unresponsive, the printers jamming, or lost wireless connectivity (needed to maintain the electronic pollbook). In these cases, the poll workers reacted (uncued, although we did provide poll worker training before the election) by making manual copies of the serial number plus marks made by the voters. Some physically signed the receipts. These could be recorded electronically later and any transcription errors would be subject to detection through the online receipt verification check.

We confirmed that the security mechanisms of Punchscan were not well understood by the voters. For example, the fact that the order of the letters on the ballots were randomized was not clearly indicated on the ballot, leaving the voter with only their intuition for forming a proper understanding of why a receipt does not contain adequate information for determining their vote. As poll workers explained this to the voters, many voters then understood. However, surprisingly, we did not observe any voters who specifically critiqued the indirection involved on the ballot. Voters did receive explicit instruction from the poll worker on this point prior to being issued the ballot and graduate students are not representative of the general population. We still advocate a removal of the indirection if possible.

Many voters indicated that the voting process was burdensome. We observed voters who appeared to not realize that they were to receive a receipt and wanted to leave immediately after marking their ballots. Furthermore, when it was explained to them that they would receive a receipt, some voters refused it. It is important for integrity that voters at least take the receipt—a left receipt means that the receipt will not be checked, opening a window of opportunity for an attacker to modify that ballot (similar information can be gathered from a garbage bin full of receipts outside the polling place). We advocate a move to opt-in auditability were voters can choose to ignore the verification aspects and there will be no discernible evidence of whether a voter constructed a receipt or not.

Another interesting finding was that approximately 85% of the voters chose to shred the top sheet and keep the bottom page as their receipt, indicating that the choice of which sheet is kept is not random (in Chapter 5, we show that voters should actually choose which sheet to keep before being issued their ballot to avoid certain coercion attacks). A single sheet ballot would resolve this issue. The ballot receipts were hosted on the Punchscan webserver and our server logs showed that the image files of 83 of the ballots were requested. Whether this means the receipt image was actually checked against the receipt cannot be determined from the available data, but it does suggest that roughly half of the voters were interested in checking their ballot receipts online.

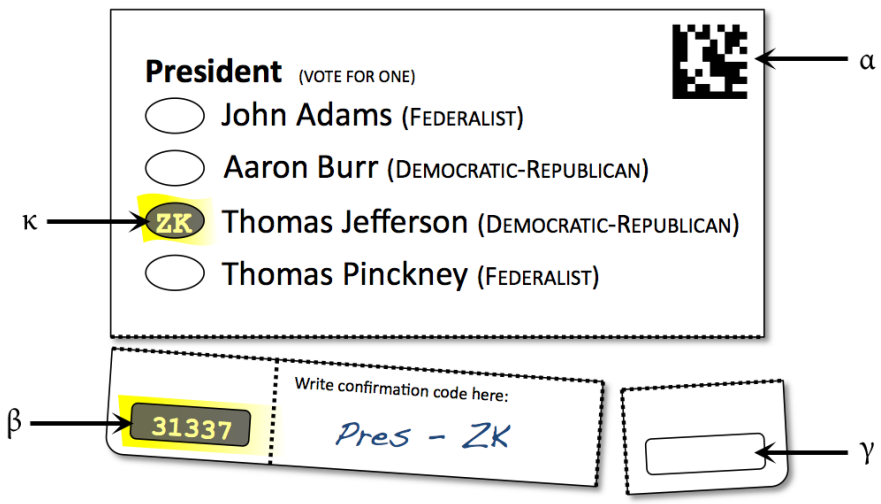


Figure 4.1: A Scantegrity ballot ensemble showing the ballot (top) with one marked position and machine-readable serial number; left receipt (bottom left) with a developed receipt serial number and confirmation code written in; and right receipt (bottom right) with an undeveloped receipt serial number. This figure is meant to demonstrate the parts of the ballot ensemble and does not represent the actual final state of the portions after voting.

4.3 Scantegrity: an E2E add-on

In this section, we consider Scantegrity, the successor to Punchscan, which is quite different in many regards from Punchscan and other human verifiable E2E systems (see Section 3.4). Scantegrity is designed foremost for practical use. This makes itself apparent in a few ways. First, it uses a single sheet paper ballot and a simple-to-perform obfuscation technique: code substitution (that is also immune to forced randomization attacks). Next, it is not designed as a replacement to existing voting systems but rather an augmentation. It can be added as a cryptographic layer to any optical scan system without interfering with the two existing methods for counting optical scan ballots: electronically from the scanned images and manually with the paper ballots. Like Punchscan, Scantegrity also trades off making certain physical security and trusted computing assumptions for increased security. We feel that many of the issues we experienced with Punchscan in the GSAÉD election are resolved in Scantegrity.

4.3.1 Ballot Features

The Scantegrity ballot ensemble is shown in Figure 4.1. It consists of two parts: a ballot and a detachable receipt. Like an optical scan ballot, the main body of a Scantegrity

ballot contains, for each contest, a list of valid selections printed in a canonical order predetermined by polling place procedures (*e.g.,* alphabetical, rotated across precincts, *etc.*). Next to each selection is a markable region, oval in shape, called a bubble. In the bubbles associated with each selection, a short alphanumeric code is printed in invisible ink that is not human readable until marked by the voter. We refer to these codes as confirmation codes. In an election with n ballots and m candidates, there are $m \cdot n$ confirmation codes. Each confirmation code is randomly drawn from a suitable space: in the example, it is two alphanumeric characters. The voter records their confirmation code(s) on the receipt and will use this information to conduct an important component of the audit.

Each Scantegrity ballot ensemble also contains three serial numbers. These are used to identify the ballot and to provide the voter with the ability to cast informational disputes under certain scenarios. The ballot contains one of these numbers in the form of a machine-readable barcode that is not easily read or memorized by a human. Since many optical scanners use marksense technology, which only records whether a region is marked or unmarked, a suitable encoding of this number must be used (*e.g.,* a data matrix for marksense scanners). For the i^{th} ballot, we refer to this serial number as α_i. The receipt portion contains two additional serial numbers, β_i and γ_i, which are printed in invisible ink and are individually detachable from the receipt.

4.3.2 Election Preparation

Prior to the election, the administrators will determine a set of officials O and the threshold of officials required, t, to conduct the election tasks. We assume that for each subset of size t, at least one official is honest. If all officials collude, they can link ballot receipts to the candidate voted for (however the integrity of the tally is invariant to collusion). The officials first generate and threshold share a secret value. This value will seed a pseudorandom generator (PRG) from which all the election data can be deterministically produced and reproduced. For each task, the election officials convene in the same room and use an attested computing platform with no static memory to generate the seed from their secret shares and to generate the election data. The computer is trusted but only for privacy. Aside from the public output produced during each session, no state is saved in any form. Every session regenerates the state from the shared secret.

In the first meeting, at least t officials input their shares along with the number of candidates, m, and the number of ballots to produce, n. The number of ballots should be larger than the anticipated number of voters since voters may generally spoil up to two ballots each and may also spoil ballots to conduct a Scantegrity-specific audit that ensures ballots are being printed correctly. The trusted workstation produces the following tables (exemplified in Figure 4.2):

Ballot ID (α)	Adams	Burr	Jefferson	Pinckney	Receipt ID (β)	Receipt ID (γ)
01	EN	PL	JG	VE	8860	3478
02	IV	QU	SA	WC	3813	2448
03	AY	EY	XW	SI	2381	3492
04	BV	SY	ZK	HJ	1337	4592
05	SH	SR	OF	XJ	3492	3472

(a) Table **P**

α	κ_0	κ_1	κ_2	κ_3	β	γ
01	JG	EN	PL	VE	8860	3478
02	QU	IV	WC	SA	3813	2448
03	AY	EY	SI	XW	2381	3492
04	HJ	ZK	SY	BV	1337	4592
05	SR	XJ	SH	OF	3492	3472

(b) Table **Q**

Q-Pointer	Marks	**S**-Pointer
$(05, \kappa_1)$		(2, Pinckney)
$(03, \kappa_3)$		(4, Jefferson)
$(02, \kappa_1)$		(4, Adams)
$(01, \kappa_3)$		(3, Pinckney)
$(01, \kappa_2)$		(4, Burr)
$(05, \kappa_3)$		(3, Jefferson)
$(04, \kappa_2)$		(3, Burr)
$(04, \kappa_0)$		(4, Pinckney)
$(03, \kappa_0)$		(0, Adams)
$(04, \kappa_3)$		(3, Adams)
$(02, \kappa_3)$		(1, Jefferson)
$(03, \kappa_1)$		(2, Burr)
$(05, \kappa_0)$		(1, Burr)
$(01, \kappa_1)$		(2, Adams)
$(02, \kappa_2)$		(0, Pinckney)
$(04, \kappa_1)$		(0, Jefferson)
$(03, \kappa_2)$		(1, Pinckney)
$(05, \kappa_2)$		(1, Adams)
$(01, \kappa_0)$		(2, Jefferson)
$(02, \kappa_0)$		(0, Burr)

(c) Table **R**

	Adams	Burr	Jefferson	Pinckney
0				
1				
2				
3				
4				

(d) Table **S**

Figure 4.2: Tables **P**, **Q**, **R**, and **S** as generated by the election officials before election day. Table **P** is kept private. The publicly published versions of tables **Q**, **R**, and **S** contain commitments to the information shown above. For example, a vote for Jefferson on ballot 04 would reveal the confirmation code ZK. The corresponding row of table **R** points to position $(04, \kappa_1)$ in table **Q** and to position $(0, \text{Jefferson})$ in table **S**.

- $\mathbf{P} := (p_{i,j})_{n\times(n+2)}$: Table $\mathbf{P}$ specifies the correspondence between confirmation codes and candidates on each ballot. Row i corresponds to ballot i and column $j \in [1, m]$ corresponds to candidate j, so that $p_{(i,j)}$ contains the confirmation code printed on ballot i and bubble j. The last two columns ($m+1$ and $m+2$) contain the receipt serial numbers for each ballot. All codes and serials are generated randomly from the PRNG. Table $\mathbf{P}$ is never published but is used to print the ballots and to generate table $\mathbf{Q}$.
- $\mathbf{Q} := (q_{i,j})_{n\times(n+2)}$: Table $\mathbf{Q}$ is the same as $\mathbf{P}$, except the confirmation codes in each row have been shuffled. If permutation π_i is applied to row i, then $q_{(i,j)} = p_{(i,\pi_i(j))}$ for $j \in [1, m]$. Thus, row i still corresponds to ballot i, but each column does not correspond to a particular candidate. The election officials commit to each confirmation code and receipt serial number in table $\mathbf{Q}$ and publish these commitments on the election website. If the commitment to a confirmation code is opened in $\mathbf{Q}$, it cannot be determined which candidate that code corresponds to (whereas if the corresponding code was opened in $\mathbf{P}$, the candidate would be known). The permutation and random factors for the commitments are generated randomly from the PRNG.
- $\mathbf{R} := (r_{i,j})_{n\cdot m\times 3}$: Table $\mathbf{R}$ contains a row for every bubble in the election (in a random order). At the end of the election, a bubble may be marked, unmarked, or selected for a print audit. Each row in $\mathbf{R}$ contains a pointer back to the confirmation code $q_{(i,j)}$ in $\mathbf{Q}$ that is printed in this bubble, and a pointer to an element $s_{(i,j)}$ in table $\mathbf{S}$, which will contain the candidate associated with the bubble. The election officials generate these pointers using the PRNG, commit to each $\mathbf{Q}$-pointer and $\mathbf{S}$-pointer, and publish these commitments on the election website.
- $\mathbf{S} := (s_{i,j})_{n\times m}$: Table $\mathbf{S}$ contains a markable region for each bubble in the election but associates the bubble with one of the candidates. Table $\mathbf{S}$ (initially empty) is published on the election website.

It may be useful to think of $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{S}$ as the information in $\mathbf{P}$ (and thus the information printed on the ballots) split across three tables. While any individual commitment has the potential of being opened under some scenario, at the conclusion of the election, only some of information in the tables will be opened to preserve ballot secrecy for voted ballots.

4.3.3 Election Day

On election day, voters arrive at the polling place, identify themselves, and registered voters are issued a ballot. Voters who are unregistered may be permitted to cast a provisional ballot, while voters with disabilities may retain the right to a fully electronic ballot. Issued ballots are locked to a clipboard by the poll-worker to prevent chain voting and the voter is directed to a voting booth.

The voter marks the ballot like an optical scan ballot, only using a decoder pen provided in the booth that reveals the confirmation codes printed in the bubble for the voter's selected candidate(s). The background of the bubble will immediately turn dark, leaving a confirmation code visible in the foreground (see Figure 4.1). The relative darkness of any marked bubbles to unmarked ones will allow an optical scanner, employing dark mark logic, to register the bubble as marked. The foreground of the bubble will be human-readable and a voter interested in participating in the election audit may record the code on the chit portion of the ballot. Uninterested voters may disregard the codes. Although not apparent to the voter, the confirmation code is printed in a slow-reacting invisible ink that will also turn dark, but only after the passage of several minutes. At this time, the bubble will be completely dark and the code will no longer be visible, leaving no human-readable unique information on the ballot. If prior to a manual recount, the data matrix serial numbers, α, are removed from the ballots, the same level of privacy as in optical scan remains.

When the voter has satisfactorily marked their ballot, it is returned to the poll worker. As previously, the poll worker unlocks the ballot from the clipboard and detaches the receipt from the ballot. Further, with the choices on the ballot still concealed, the poll worker places the main body of the ballot into the scanner, which records the ballot serial number, σ, and the marked choices, κ's. If the scanner reports an undervote or overvote, the voter may opt to spoil the current ballot and re-enter the issuance procedure, or they may proceed in casting the ballot if the polling place procedures allow.

After a successful scan, the two serial numbers on the receipt, β and γ, are developed by the poll worker and the voter may leave with the receipt. It is expected that public interest groups will make available the possibility of creating a copy of receipts to alleviate the need for concerned but time-constrained voters to personally participate in auditing the election.

Voters may also opt to check that the ballots are printed correctly. In order to do this, the voter requests a ballot to be print audited, reveals all the confirmation codes on the ballot, and chooses one of the receipt serial numbers, β or γ, to retain with the codes. The election authority can keep the other code as a record of both the ballot that was audited and the code which was kept by the voter.

If the voter makes an error in marking their ballot or wishes to register a protest vote through spoiling their ballot, the ballot is returned to the poll worker. Without seeing the contents of the ballot, the poll worker unlocks the ballot from the clipboard and detaches the right side of the receipt, β, from the ballot. The main body and left receipt, α and γ, are shredded in view of the voter. The right receipt is retained by the poll worker and used in balancing the number of ballots issued with the number of ballots tallied, print audited, or spoiled.

4.3.4 Posting and Tallying

After the close of polls, the election officials publish a list of all the voters who voted and the tally given by the underlying optical scan system. The electronic ballot images from the scanner, the list of audited ballots, and list of spoiled ballots are entered into the trusted workstation by the trustees. Table **P** is regenerated and used to translate the votes into the confirmation codes, which were revealed on the cast ballots. The commitments in table **Q** to the confirmation codes and serial numbers that were revealed during the election are opened, and corresponding marks are posted in **R** and **S** (exemplified in Figure 4.3). Anyone can now compute the tally from **S**. This tally is checked against the one reported by the optical scanners or manual recount.

Once the results are posted, a voter who has made a receipt of her confirmation codes can go to the election website and look up her voted ballot by either receipt serial number. She checks that, in the row of **Q** corresponding to her ballot, all and only her confirmation codes appear. Anyone can check that the opened commitments match the confirmation codes on the election website. If any of the confirmation codes from the voter's receipt does not appear posted in **Q**, the voter should file a dispute.

For those ballots that were chosen for a print audit, election officials open the commitments to all the information associated with the ballot, *i.e.*, the confirmation codes in **Q** and the **Q**-and **S**-pointers in **R**.

4.3.5 Auditing the Results

To verify that the marks were added correctly to tables **R** and **S**, the election officials will be challenged to open either the **Q**-pointer or the **S**-pointer in table **R**; *i.e.,* the first or third column (exemplified in Figure 4.4). If a mark was recorded for the wrong candidate, then either (i) the mark is not consistent with the corresponding mark in **R**, (ii) the marks in **R** and **S** match but the status of the corresponding confirmation code (revealed or unrevealed) in **Q** is inconsistent, or (iii) the tables are consistent but the revealed code in **Q** does not match the receipt. The receipt check probabilistically detects (iii), while the present challenge will detect either (i) or (ii). The challenge is generated with a publicly verifiable random number (see Chapter 6).

Any interested party can check that the commitments are correct: that each revealed **Q**-pointer in table **R** either connects a revealed code in table **Q** to a marked element in table **R** or connects a hidden code to an unmarked element, and that each revealed **S**-pointer in table **R** either connects a marked element in table **R** to a marked element in table **S** or connects an unmarked element to an unmarked element. Essentially, the audit checks that marks are mapped unchanged from table **Q** through table **R** to table **S**.

α	κ_0	κ_1	κ_2	κ_3	β	γ
01			PL		8860	3478
02				SA	3813	2448
03	AY	EY	SI	XW		3492
04		ZK			1337	4592
05			SH		3492	3472

(a) Table **Q**

	Adams	Burr	Jefferson	Pinckney
0	A		✓	
1	✓		✓	A
2		A		
3				
4	✓		A	

(b) Table **S**

Q-Pointer	Marks	**S**-Pointer
$(03, \kappa_3)$	A	$(4, \text{Jefferson})$
	✓	
$(03, \kappa_0)$	A	$(0, \text{Adams})$
	✓	
$(03, \kappa_1)$	A	$(2, \text{Burr})$
	✓	
$(03, \kappa_2)$	A	$(1, \text{Pinckney})$
	✓	

(c) Table **R**

Figure 4.3: Tables **Q**, **R**, and **S** as published after the close of the election. For each revealed confirmation code in table **Q**, the row corresponding to that code in table **R** and the element corresponding to that code in table **S** have been flagged. For each row of table **R**, either the commitment to the **Q**-pointer or the commitment to the **S**-pointer has been opened and published. Note that the revealed confirmation codes in table **Q** (other than those for ballot 03, which is a ballot chosen for a print audit), the rows flagged in table **R**, and the flags in table **S** are in one-to-one correspondence.

Q-Pointer	Marks	**S**-Pointer
$(05, \kappa_1)$		
$(03, \kappa_3)$	A	$(4, \text{Jefferson})$
$(02, \kappa_1)$		
$(01, \kappa_3)$		
$(01, \kappa_2)$	✓	
$(05, \kappa_3)$		
$(04, \kappa_2)$		
$(04, \kappa_0)$		
$(03, \kappa_0)$	A	$(0, \text{Adams})$
$(04, \kappa_3)$		
$(02, \kappa_3)$	✓	
$(03, \kappa_1)$	A	$(2, \text{Burr})$
$(05, \kappa_0)$		
$(01, \kappa_1)$		
$(02, \kappa_2)$		
$(04, \kappa_1)$	✓	
$(03, \kappa_2)$	A	$(1, \text{Pinckney})$
$(05, \kappa_2)$	✓	
$(01, \kappa_0)$		
$(02, \kappa_0)$		

(a) Table **R** (Heads)

Q-Pointer	Marks	**S**-Pointer
		$(2, \text{Pinckney})$
$(03, \kappa_3)$	A	$(4, \text{Jefferson})$
		$(4, \text{Adams})$
		$(3, \text{Pinckney})$
	✓	$(4, \text{Burr})$
		$(3, \text{Jefferson})$
		$(3, \text{Burr})$
		$(4, \text{Pinckney})$
$(03, \kappa_0)$	A	$(0, \text{Adams})$
		$(3, \text{Adams})$
	✓	$(1, \text{Jefferson})$
$(03, \kappa_1)$	A	$(2, \text{Burr})$
		$(1, \text{Burr})$
		$(2, \text{Adams})$
		$(0, \text{Pinckney})$
	✓	$(0, \text{Jefferson})$
$(03, \kappa_2)$	A	$(1, \text{Pinckney})$
	✓	$(1, \text{Adams})$
		$(2, \text{Jefferson})$
		$(0, \text{Burr})$

(b) Table **R** (Tails)

Figure 4.4: Table **R** as published after the close of the post-election. According to a coin-flip, either the entire the correspondence between **Q** and **R** is revealed, or the entire correspondence between **R** and **S**. In an actual election, many independently permuted copies of **R** will be generated and can be audited in this fashion.

Currently the soundness of the check is only $1/2$. However by generating and using x independent **R** tables throughout the election, each with a unique random mapping from **Q**, the soundness increases to $1 - 1/2^x$ through auditing each independently. This is effectively the cut-and-choose protocol proposed by Sako and Kilian [SK95]. Alternatively, each row in each **R** can be individually challenged to be opened either way, which is effectively randomized partial checking by Jacobsson, Juels and Rivest [JJR02]. In the latter case, voter anonymity is split into two anonymity sets, and care must be taken to randomize both the mapping from **Q** into **R** and the mapping from **R** into **S** to prevent further intersection of the anonymity sets.

4.3.6 Dispute Resolution

Dispute resolution is one of the key features of Scantegrity. Many E2E systems do not adequately consider how disputes might be settled. If a dispute cannot be resolved, then a default position must be taken, which will generally either allow a corrupt election authority to get away with fraud or allow a malicious party to prompt a reelection (or at least cast doubt on the result) by filing spurious disputes.

To file a dispute, the voter must be on the registration list as being eligible to vote or as having cast a ballot, and only one dispute may be filed per voter. (If the voter wishes to allow a third party organization to check her receipt, she might sign over the right to her dispute.) Disputes are filed within a period of time. After the period closes, the trustees open **Q** completely, revealing all the confirmation codes and receipt numbers on all ballots (except spoiled ballots), and use this information to address the disputes.

The most likely dispute is that a voter's confirmation code does not match the one appearing on the website. In this case, the voter provides the ballot ID and a claim of what the correct confirmation code(s) should be. Disputes of this type could be (i) the result of the voter making a transcription error, (ii) a mischievous voter attempting to call into question the legitimacy of the election, (iii) an error by the scanner, or (iv) evidence of fraudulent behaviour.

With a transcription error, the claimant's code is likely close to the revealed code. Once **Q** is opened, the voter can verify that none of the other codes that appeared on the ballot match her code exactly. Since voters do not see the confirmation codes of the unrevealed candidates on the ballot when they file, the probability that a code that a voter claims to have received is actually one of the other codes committed to for her ballot is very small if the voter made a transcription error or is merely guessing. The election officials eliminate from consideration any disputes for which none of the opened codes match the claimed code. We consider the remaining disputes to be plausible discrepancies.

Plausible discrepancies could still be the product of cases (i) or (ii); however, the

probability is small (and can be made smaller by increasing the length of the confirmation codes). They are more likely (iii) or (iv). The election officials should set up a statistical trigger, based on various election parameters, such that when a given number of plausible discrepancies is reached, the ballots are rescanned to rule out (iii) and then the election is considered to have strong evidence of fraud.[3]

Another type of dispute that a voter may have is that her ballot is improperly designated as voted, print audited, or spoiled. A voter who cast her ballot knows both receipt serial numbers $\{\beta_i, \gamma_i\}$. If a ballot is represented as print audited or spoiled, either one or neither of these codes will be open in $\mathbf{Q}$. Similarly the voter retains evidence that a ballot was print audited by knowing all the confirmation codes on the ballot. If a ballot is represented as voted or spoiled, either one or none of the confirmation codes will be open in $\mathbf{Q}$. If the voter's claimed receipt code(s) or full set of confirmation codes match their committed values, the dispute is designated as a plausible discrepancy.

4.4 Scantegrity Case Study

In November 2009, we employed Scantegrity in its first binding election. This marked the first time an E2E system was used for in-person voting in a governmental election.[4] The election was run by the city of Takoma Park, Maryland, USA to elect city councillors and a mayor.

4.4.1 Requirements

The election consisted of a single polling station, although the election itself was broken into six wards. The mayor race was common to all wards and the councillor race was specific to the ward. Voters were issued one of six ballot styles based on their ward. The city has approximately 17,000 residents and 10,934 voters were registered. The turnout was 1,728 voters.

Demographically, the median household income in Takoma Park in 2004 was $48,675. The percentage of households with computers was 87.4%, and about 32% of Takoma Park residents above the age of twenty-five had a graduate, professional or doctoral degree.[5]

[3] If this were to happen, it would hopefully trigger litigation (as well as a criminal investigation). In Canada, any elector or candidate may apply for a contested election proceeding and if heard, the judge reserves the power to invalidate the election result (upon appeal to the Supreme Court of Canada).

[4] We generally believe it to be the first use of any kind of E2E system, however one could argue that, for example, the remote voting system RIES, used in the Netherlands in 2004 and 2006, was E2E. In principle, it may have been by design, but the implementation provided effectively no ballot secrecy and had numerous problems, and the project was subsequently cancelled [Gro04b, HJS+08, GHH+09].

[5] The Montgomery County US Census Update Data, 2005

The latter are above-average for US cities, and are favourable toward a random sample of citizens understanding, accepting, and finding Scantegrity usable. However there is debate in the political science literature as to how representative voters are of the general population, and so a notable shift from the mean in the demographics of the general population may not translate into the equivalent shift in the voting population (or it may be amplified).

Ballot. The ballot was required to be worded in both English and Spanish. A bilingual ballot was used instead of different sets of ballots for each language. For each ward, the ballot consisted of two contests. The joint mayor contest contained two named candidates and a write-in candidate. The ward councillor races varied between one and two named candidates and a write-in candidate for each.

Write-in candidates. For each contest, voters could elect to write-in a candidate. In Scantegrity, write-in candidates are considered a single candidate. The candidate has a confirmation code and all write-in votes are grouped together in the initial tally. If the number of write-ins could be relevant to determining the winner (either of the election or the round in a multi-round scoring protocol), the actual candidates voted for are then examined. In this election, write-ins were not factor in any of the contests.

Instant Runoff Voting (IRV). Takoma Park has used IRV in municipal city elections since 2006. IRV is a ranked choice system where each voter assigns each candidate a rank according to her preference. Prior to using Scantegrity, Takoma Park's IRV ballots were arranged with a matrix for each contest, with candidates on the rows and preference order on the columns. We adapted this style and put confirmation codes in each cell.

Absentee ballots. Takoma Park offers the option of casting a ballot by mail. While we believe the invisible ink technology we use is secure against any reasonable attack that could be mounted in a polling booth, we were reluctant to provide ballots to absentee voters who could use specialized equipment to read all the codes on the ballot, and then file spurious disputes that would cast doubt on the election result. The city was also reluctant to mail decoder pens to all the absentee voters. We agreed to use Scantegrity ballots without the confirmation codes. This means absentee voters cannot verify the codes (although they can verify that their ballot was received). 70 absentee ballots were cast.

Voter registration. Registration was handled by the city of Takoma Park. Scantegrity did offer the option of provisional ballots, which would allow voters to vote but the ballots

would be escrowed until the voter's eligibility was confirmed. If confirmed, the ballots are added to the tally (this itself may break the secrecy of the ballot if the set of approved ballots is small, however this is a known issue in any election system providing provisional voting).

Disclosed source code. All the software used in the election—for ballot authoring, printing, scanning and tally—was published well in advance of the election as commented, buildable source code, which may be a first in its own right. Moreover, commercial off-the-shelf scanners were adapted to receive ballots in privacy sleeves from voters, making the overall system relatively inexpensive.

Election authority. It was decided that the election would have four trustees (the chair, vice chair and a member of the Board of Elections, and the city clerk) and any two were required to regenerate the election data.

4.4.2 Mock Election

In April 2009, we held a mock election to test and demonstrate feasibility of the Scantegrity system, as well as train the poll workers. The mock election was held during Takoma Park's annual Arbor day celebration at the city hall. Volunteer voters, recruited from people attending the celebrations, voted on a mock ballot with questions relating to trees. Turnout was 95 voters. The election system used in the mock election was very close to the theoretic system we described in the previous section. We uncovered several minor issues and one main issue: the average time-to-vote was unacceptably high at 8 minutes.

We identified two main impedances to voter flow. The first was filling out the ballot. The mock ballot had two IRV questions with four and five candidates, and two single response question. A fully marked ballot would consist of marking 11 bubbles and recording 11 confirmation codes. We reasoned that the election ballot would be quicker to fill out because the ballot was much shorter (at most 6 bubbles/codes) and voters would likely have a preconceived notion of who to vote for.

The second impedance was the scanning station. During the mock election, voters would have their ballots scanned, be shown a representation on a monitor screen of what was scanned, and manually click to accept the interpretation and electronically cast the ballot. Then the receipt would be detached and the receipt codes would be revealed by the poll worker. Finally, the ballot was deposited in the ballot box. In the real election, we eliminated the verification step. Ballots were deposited into a slot, which resulted in the ballot being scanned and deposited into the ballot box. We removed the receipt from the ballot itself and instead provided separate verification cards. Voters who wanted to

verify their ballot could copy the ballot serial number and confirmation codes onto the card.[6] Additionally, we eliminated the receipt codes from the ballots. Instead of using them to provide dispute resolution, we relied on the assumption that the poll book would correctly maintain the number of cast, audited and spoiled ballots, as well as the original copies of any spoiled or audited ballots (for audited ballots, a photocopier was provided). Finally, we decided to use a second scanning station in the actual election. Through these modifications, the average time-to-vote in the actual election was reduced to under 3 minutes.

In terms of minor issues, we replaced the pens with dual-ended pens. The one end had a chisel tip appropriate for marking ovals and the other had a fine-tip, appropriate for writing in candidates or writing down codes. We also improved the printing to eliminate some smudging of the codes. In the mock election, we used locks to prevent chain voting like we did in the Punchscan election however these were unpopular with the poll workers and eliminated. Last, we changed the confirmation codes from two letters (from a reduced set eliminating easily confused characters) to three decimal digits. In retrospect, we should have eliminated 0 and 8 from the set of digits.

4.4.3 The Election

The preparation of the election followed essentially the same steps as the Punchscan election, so we eliminate some details. The ballots were designed to closely resemble the ballots used in Takoma Park's previous municipal election. It contained some instructions for marking an IRV ballot and for marking and verifying a Scantegrity ballot.

Ballot printing. We had the invisible ink (and developing ink and pens) manufactured for our use. The ballots were printed on off-the-shelf CMYK inkjet printers. We kept the black ink in the CMYK printer cartridge and replaced the other 3 inks with the invisible ink, a dummy ink, and a florescent ink. The invisible ink is actually visible as a light yellow but turns dark grey when developed with the pen. The dummy ink is the same yellow colour but does not develop. We print it around the invisible ink to make the inks indistinguishable. Finally, we print a random scatter pattern over each bubble with the florescent ink to further obfuscate attempts at reading the undeveloped codes. Over many months, the invisible and dummy inks oxidize differently and the codes can become visible but this is not an issue if the ballots are printed shortly before the election.

[6] This also makes it difficult to determine if a voter intends to verify their ballot or not. Recall in the Punchscan election, we had the difficulty of uninterested voters refusing their ballot-specific receipts or discarding them at the polling place.

Voter Education. Articles in the City newspaper before the election introduced the verification mechanisms provided by Scantegrity and explained how voters could check their confirmation codes online. This also appeared on the city's election website. During the election, a local radio station covered the election and the use of Scantegrity, prompting many curious voters to attend the polling place. Finally, the ballot itself contained some instruction. Voter education was identified as a main area to improve in future elections.

Election day setup. The scanning station was the only Scantegrity-specific equipment. It consisted of a netbook, located in a lockbox, to control the scanner. Upon booting, the scanning software ran automatically off an attested, read-only SD card in the netbook. The election data was written onto USB sticks. Scantegrity is software independent and will detect if the scanner misbehaves in a way the effects the tally, however we strived to prevent this to avoid having to rerun aspects of the election. The scanner itself was embedded into the form factor of the scanning station. An uninterruptable power supply was also used in the case of a power failure.[7]

Voting. Polls were open from 7 am to 8 pm. Ballots were issued to registered voters in a privacy sleeve. Voters would mark their ballots in the voting booth, optionally fill out a verification card, and be directed to the scanning station. At the scanner, voters simply dropped the privacy sleeve (with the ballot in it) into a slot. A rivet in the privacy sleeve prevented it from reaching the scanner, while the ballot itself slipped out and into the scanner's intake. The bottom of the scanner was inside the ballot "box" (which was actually a large bin on wheels making the scanner at waist height). The privacy sleeve could be removed from the slot and returned to the ballot issuing station by the poll workers.

Print audits. During the day, an independent auditor from EPIC would visit the polling place at an unannounced time and select a random set of ballots. All the codes on these ballots were revealed and the auditor retained a copy of them for conducting a print audit against the cryptographic information.

Tallying. The trustees used the Scantegrity software to generate the tally at 10 pm. The Chair of the Board of Elections (and one of the trustees) announced the results to those present at the polling place at the time (including candidates, their representatives, voters, etc.); this was also televised live by the local TV station. Confirmation codes and the

[7] If use of the scanner was somehow lost, the Scantegrity election could still proceed without a problem. Ballots would be collected and scanned centrally later. The confirmation codes provide a cryptographic chain-of-custody over the ballots.

election day tally were posted on the Scantegrity website. The tally was updated the next day with votes from approved provisional ballots, which did not change the result.

Hand Count and Certification. Members of the Scantegrity team and the trustees conducted a hand count of the ballots and certified the results. The hand count and the Scantegrity count differed slightly because officials were able to better determine voter intent during the hand count. For example, in the mayoral race, the scanner count determined that 646 votes were cast for candidate Schlegel, 972 for Williams, 15 for various write-in candidates, and 90 were not cast. The certified hand count totals were 664 votes for Schlegel, 1000 for Williams and 17 for write-in candidates. Thus 48 of a total of 1681 votes in this race would not have been counted by a scanner count alone. The discrepancy was caused by voters marking ballots outside of the designated marking areas. Such marks, while not read by the scanner by definition, are considered valid votes by Takoma Park law. Similarly, 8 of a total of 447 votes for Ward 1 council member, 8 of 251 for Ward 2, 16 of 431 for Ward 3, 10 of 210 for Ward 4, 2 of 81 for Ward 5 and 11 of 199 for Ward 6 were added to scanner vote totals after hand counting.

Dispute Resolution. On November 6, the dispute resolution period ended. Scantegrity received a single complaint by a voter who had trouble deciphering a digit in the code and noted it as "0," while the Scantegrity website presented it as "8." The voter requested that codes be printed more clearly in the future. He also stated that if he were not a trusting individual, he would believe that he had proof that his vote was altered.

Post-Election Audit. On November 6, trustees used the Scantegrity software to conduct the cut-and-choose audit (with a public challenge computed from stock market data). By November 9, two independent auditors retained by Takoma Park (Ben Adida and Filip Zagórski—both cryptographic voting researchers whose work we covered in Chapter 3) had verified the results with auditing scripts they independently wrote and published based on the Scantegrity protocol. The auditor from EPIC (Lillie Coney) later verified that all the ballots she collected were printed correctly.

Receipt Check. The Scantegrity website recorded 81 unique ballot ID verifications, of which about 66 (almost 4% of the total votes) were performed before the dispute resolution deadline. We also were informed that at least a few voters had checked their codes through one of the independent auditor's websites, both of which had made the confirmation codes available. The number of voters who checked their ballots, while not large, was sufficient

to have detected (with high probability) any errors or fraud large enough to have changed the election outcome.[8]

4.4.4 Lessons Learned

Voter observation. To understand the experiences of voters and poll workers, observers from UMBC timed some of the voters as they voted, asked voters and poll workers to fill out two questionnaires, and informally solicited comments from voters as they left the precinct building. Based on a sample of 93 voters, voting times ranged from 55 seconds to 10 minutes with a mean of 167 seconds. Most of the time was spent marking the ballot. The observers also noted that two voters inserted the ballot and privacy sleeve into the scanner upside-down, causing the ballot not to be fed into the scanner. Others had difficulty inserting the sleeve with a single hand. The usability of the privacy sleeves and their interaction with the scanner could be improved.

Voter survey. As voters were leaving the precinct, members of the Scantegrity team invited them to fill out a survey that we had designed. The survey included questions about demographics and their experiences voting. Most answers were expressed on a seven-point Likert scale and voters were also invited voters to make any additional comments or suggestions. 271 voters participated in the survey.[9]

Figure 4.5 shows how voters responded to four questions from the questionnaire. These results strongly show that voters found the voting system easy to use and that they had confidence in the system. We also learned that providing the option to check receipts online increased voter confidence in the election results. While the receipt check is a necessary condition for E2E verification, it is not sufficient, so it is open to further investigation whether voters understand the distinction. Finally we determined that voters had confidence that the receipt alone did not reveal how they voted; this finding is notable given that it is widely suspected that many people erroneously believe that all E2E receipts reveal ballot choices.

A significant minority of the surveyed voters responded very negatively to the system. Without running a control experiment, with voters casting votes on a normal optical scan system, we may never conclusively isolate the negative (and positive) reactions to Scantegrity itself from reactions to electronic voting in general or the instant run-off scoring protocol used. Of the voters surveyed, 51 provided additional comments and suggestions which shed some light on the negative reactions.

[8] We omit detailed calculations, noting these calculations become quite complex due to the use of IRV.

[9] A paper with a more comprehensive analysis of the results, tentatively titled "Exploring Reactions to Scantegrity: Analysis of Survey Data from Takoma Park Voters and Election Judges" is under preparation.

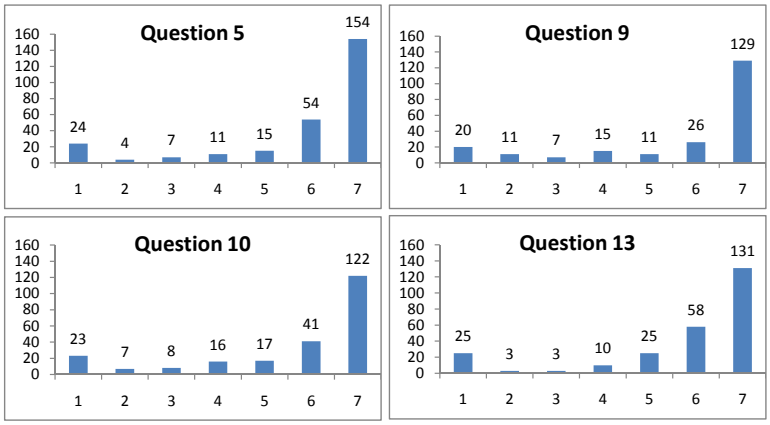


Figure 4.5: Voter responses to Survey Questions 5, 9, 10, 13 from all 271 voters completing the survey. Using a seven-point Likert scale, voters indicated how strongly they agreed or disagreed with each statement about the voting system they had just used (1 = strongly disagree, 7 = strongly agree). Each histogram shows the number of voters responding for each of the seven agreement levels. The four questions shown are the following: (5) Overall, the voting system was easy to use. (9) I have confidence that my receipt by itself does not reveal how I voted. (10) The option to verify my vote online afterwards increases my confidence in the election results. (13) I have confidence in this voting system.

The comments suggested that some voters were unaware of the verification option and did not realize they were supposed to write down the confirmation codes. Some voters did not realize that the developer pens would expose confirmation codes on the ballots, while others were confused about which end to use. Once the codes were exposed, some voters noted that the confirmation codes were hard to read. Some voters did not know how to place the ballot into the scanner. One voter had no difficulty but wondered if seniors or people who speak neither English nor Spanish might have difficulties. Another voter wondered if the government might be able to determine his vote by linking his IP address, used during the receipt check, with his ballot serial number and noting the time that he was issued a ballot. Many suggested that it would have been helpful to have better instructions, including instruction while they wait in line. Other comments were not specific to Scantegrity: some addressed instant-runoff voting or electronic voting in general.

Poll worker survey. The team also provided a survey to the poll workers, which they could return to us by mail in a self-addressed stamped envelope. Of the 12 poll workers, 5 participated. They noted that some voters did not understand what to do, including how to create a receipt or mark an instant-runoff ballot. They observed that the privacy sleeve was hard to use with one hand and that the double-ended pens created confusion. One wondered if Scantegrity was worth the extra trouble. They offered several suggestions:

simplify the ballot, provide automated receipts so that voters do not have to copy the confirmation codes, and develop better pre-election voter education.

Automated receipts. One active area of research within the Scantegrity team is the integration of a receipt printer with the system. This would allow voters to retain a copy of their receipts without having to manually fill it out. In the case of print audits, where every code needs to copied (and usually for multiple ballots), this could be a vast improvement. Integrating it has its complications however. It could be a dumb component that does not know the codes and just prints images of the revealed codes (but would have to obfuscate their position on the original ballot), or it could be smart and know what the codes should be, however this is secret information in the election.

4.5 Open Problems

In the remaining chapters within Part I, we examine in detail several issues that intersect with Punchscan, Scantegrity, and the elections we ran with both.

Coercion contracts. Although we did not realize it during the GSAÉD election, there exists a subtle coercion attack against the Punchscan ballot casting process. We analyze it in Chapter 5 and show that a simple reordering of the ballot casting process rectifies it. While discussing Scantegrity and the Takoma Park election in this chapter, we outlined certain procedures for spoiling ballots. As we will discuss in Chapter 5, these procedures were carefully designed to harden Scantegrity against the same type of attack.

Random beacons. While discussing both the GSAÉD and Takoma Park elections, we referenced the fact that stock prices were used to generate the cryptographic challenges used to audit the election. While we felt this approach was sound, we had not rigorously examined it. This led to one issue in the Takoma Park election where the volume of a stock we had used in the challenge changed between the time we used it, very soon after the market had closed, and when the independent auditor reconstructed the challenge a few days later. In Chapter 6, we provide a detailed study of generating publicly verifiable random challenges from financial data, showing the approach is sound within computational finance models.

Election verification. We were grateful to have distinguished independent auditors produce software tools, in Python and Java, to audit the Takoma Park election. However when the present author and his Chapter 7 co-author tested the tools, we experienced

difficulty executing them as they required specific versions of Python and the Java JDE respectively (as well as external libraries). In Chapter 7, we present a new verification interface called Eperio that has many benefits, including the ability to conduct the audit in a spreadsheet without any custom tools. We also provide a Python implementation, however it is designed specifically to run out-of-the-box on standard Linux and OS X distributions, including Linux Live CDs. Our goal is to reduce the barrier to performing the post-election audit in E2E systems for more voters.

Untrustworthy printing. In both the GSAÉD and Takoma Park elections, a trusted printer was needed to produce the ballots. Since printing the ballots also required specialist knowledge (special drilled paper for Punchscan and invisible inks for Scantegrity), the printing was handled by us. This introduces a chain-of-custody assumption from the workstation outputting the files to be printed through to printing the ballots. For Punchscan, this extends into the election. In Scantegrity, because of the invisible ink, once printed, reading the confirmation codes would involve developing the ink and be evident. In Chapter 8, we sketch a two-party protocol for distributing the printing of a secret between two non-colluding printers. While this approach still needs to be fully integrated into an E2E system, it shows that in principle, this one remaining chain-of-custody assumption can be removed.

4.6 Concluding Remarks.

In 2005, just after the introduction of the first human verifiable E2E voting systems by Chaum [Cha02, Cha04] and Neff [Nef04], a list of recommendations were proposed by Karlof, Sastry and Wagner for realizing the full deployment potential of cryptographic voting techniques. To summarize, they recommended the following:

- **Certification**: a new framework for evaluating E2E systems and criteria for their certification by an independent body.
- **Usability evaluation**: review of the systems by usability experts and running trials with questionnaires.
- **Recoverability**: the ability to fallback to a reliable underlying system, such as hand-countable paper ballots.
- **Transparency**: full disclosure of source code and documentation of the systems.

In 2005, these were all open problems. Through our work, and the work of others in the community, progress has been made on all four.

Certification. In the United States, the Election Assistance Commision (EAC), with guidance from the National Institute for Standards and Technology (NIST), provides vote system guidelines [Uni09]. These guidelines now include E2E voting systems. In 2010, NIST held a workshop on E2E voting systems and we reported our experiences with the mock election at Takoma Park (the workshop preceded the actual election).

Usability. While independent usability evaluation of all E2E systems is significantly deficient from the literature, the data we collected after the Takoma Park mock and municipal elections currently represents the largest dataset of voter and poll-worker reactions from an E2E system.

Recoverability. The introduction of Scantegrity, which simply added E2E verification to an otherwise normal optical scan system, represents a major step forward for recoverable E2E systems. Most paper-based E2E systems are like Punchscan, without a recoverable paper audit trail. This has benefits from a privacy perspective—the E2E system is the only interface to the real tally (even the scanner does not know how you voted). But it also creates a concentrated point of failure. We believe systems like Scantegrity are a first step toward verifiability, and increased voter privacy should follow after time.

Transparency. Both Punchscan and Scantegrity are fully open source projects (although they license patented technology; see disclosure below). In fact, to our knowledge, the Takoma Park election also has the distinction of being the first open source government election held in the United States.

The successful E2E voting pilot at Takoma Park demonstrates that voters and election officials can use advanced cryptographic techniques within an election, and, with reference to our polling data, be satisfied with its usability. We believe the Scantegrity system and this election demonstrate a significant advancement in the technical maturity of E2E voting. We can now say it is ready for real binding governmental elections. The remaining hurdle for the acceptance of cryptography in elections is when voters stop asking why we are using it, and start demanding why we are not using it.[10]

[10] Adapted from Stu Feldman's roadmap for technical maturity (as quoted in [Gee01]): (i) You have a good idea. (ii) You can make your idea work. (iii) You can convince a gullible friend to try it. (iv) People stop asking why you are doing it. (v) Other people are asked why they are not doing it.

4.7 Disclosure

Since its inception, and until the date of this publication, Scantegrity has been a *pro bono* initiative. This election was financed by the members of the Scantegrity team through research grants or personal contributions, and the team received no compensation from the City of Takoma Park. Portions of the Scantegrity system may be covered by pending patents under applications US 2008/0272194, and US 2009/0308922. All source code was released under the GPLv2 software license.[11]

[11] `http://www.gnu.org/licenses/gpl-2.0.html`

Chapter 5

A Game-Theoretic Analysis of Coercion Contracts

This chapter is adapted from published work supervised by Urs Hengartner and Kate Larson [CHL09].

5.1 Introductory Remarks

End-to-end verifiable election systems (E2E systems) allow voters to independently verify the correctness of the final tally, without needing to trust the chain-of-custody over the ballots after the election in paper voting settings, nor any software or hardware used for vote capture and tallying in electronic and remote voting settings. E2E systems often use cryptographic primitives to achieve these properties while maintaining the secrecy of every cast ballot. A sample of recently proposed E2E systems can be found in Chapter 3.

A common element of these systems is the production of some kind of obfuscation of each vote, which voters can retain, digitally or physically, as a privacy-preserving receipt of their vote. Since the receipt does not reveal which candidate the voter selected, it ostensibly cannot be used effectively in a scheme to buy votes or coerce voters into voting for a particular candidate. However this is not the case: receipts can offer ballot secrecy but not be coercion-resistant (see Section 2.2.1). Even if votes are correctly obfuscated, undue influence can still be accomplished by paying or forcing voters to follow certain procedures in the construction of their receipts, such that the receipts become probabilistically biased toward a chosen candidate. We call these procedures, and consequences for not following them, a contract. In this chapter, we argue that contracts are persistent enough in E2E systems to warrant further study and, in response, we conduct a detailed analysis in a representative E2E system—Punchscan.

Contributions. The contributions of this chapter can be summarized as:

- a new analysis of the effectiveness of three existing attacks [BMR07, KRMC10, MN07] using coercion contracts in Punchscan with two candidates,
- a definition of optimality for contracts and a linear-time algorithm for generating optimal contracts,
- an analysis of multiple-candidate contracts showing that their effectiveness decreases with the number of candidates,
- an analysis of contracts in the setting where some voters have intentions other than accepting the highest payment available to them and hide their real intentions from the adversary, and
- an analysis of contracts in the setting where the adversary is financially constrained showing that the adversary must value the vote by, approximately, an order of magnitude more than the voter selling the vote.

We are interested in cases where given only an obfuscated vote, the voter's selection remains hidden; yet if certain decisions in the construction and verification of the obfuscated vote are dictated to the voter by an adversary, the voter's compliance results in a non-negligible probability that the voter selected the adversary's preferred candidate.[1] We call such a set of instructions a *contract* and this class of attack *contract-based attacks*.

Contract-based attacks have been proposed for a variety of E2E systems. They have also proven non-trivial to avoid in the design of Scantegrity, which has been specifically hardened against them (see Chapter 4). It is our belief that this category of attack is sufficiently widespread that a detailed analysis of contracts can provide value to election system designers in understanding the mechanisms at play and the effectiveness of these attacks in a realistic setting. Instead of a light-touch on a range of systems, we have undertaken a very detailed analysis of contract-based attacks in one representative system—Punchscan [PH06]—which has been found to be vulnerable in this regard [BMR07, KRMC10, MN07]. A description of the Punchscan system can be found in Section 3.4.3.

[1] Other types of manipulation may include forcing the voter to cast a random vote [JCJ05] or to vote *against* a particular candidate instead of *for* one. This latter distinction is called destructive manipulation, as opposed to constructive, and can be accomplished through a combination of constructive manipulations. Forming a strategy of constructive manipulations can be intractable in the worse-case for some scoring protocols but it is trivial for plurality voting [BO91].

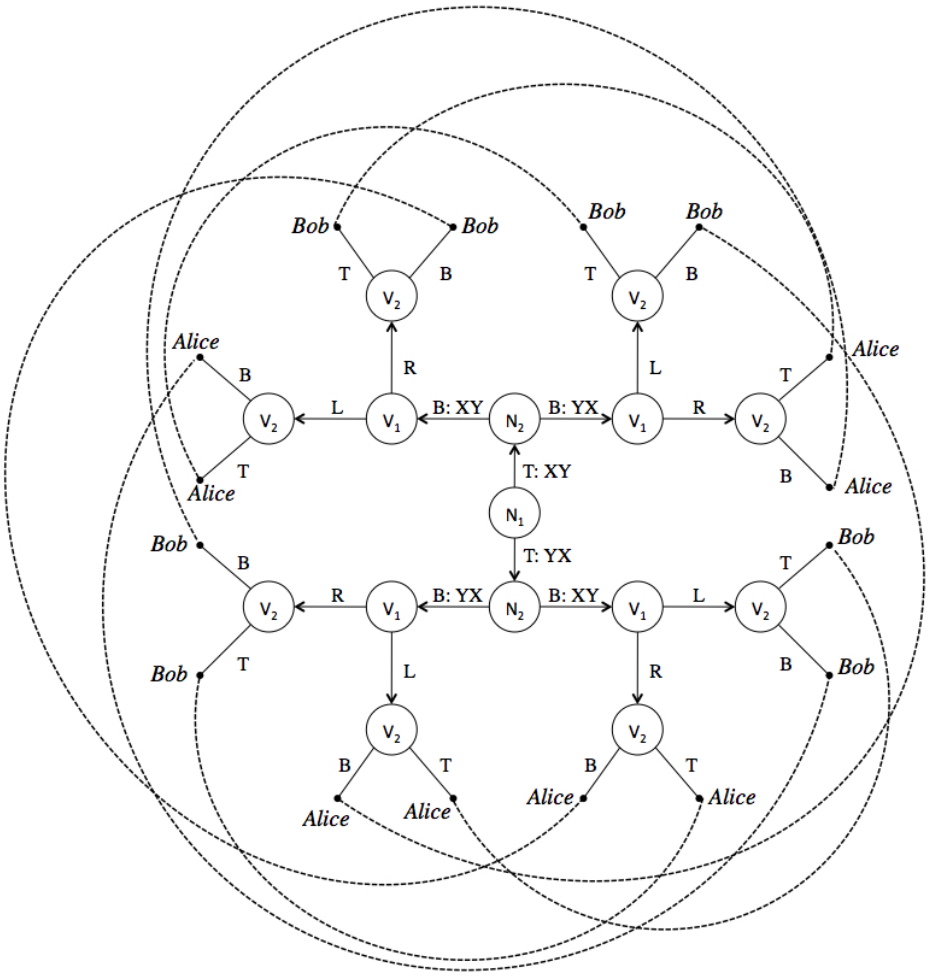


Figure 5.1: Extensive form of the ballot casting process, involving the voter and nature, in a Punchscan election. The ballot casting process begins at N_1 and ends at a candidate. The dotted lines represent hidden information.

5.2 Extensive Form of the Ballot Casting Process

To analyse the effective privacy of Punchscan ballot receipts, we will formalize the ballot casting process using game-theoretic conventions.[2] Contract-based attacks will ultimately involve three players—nature (N),[3] the voter (V), and the adversary or influencer (I),

[2]All game theoretic conventions employed in this chapter can be found in most introductory textbooks on the subject (*e.g.,* [Osb03]). Future footnotes will provide additional background on game theoretic concepts as they are used.

[3]When a game incorporates randomness, a special player called nature chooses random actions from a known distribution as needed.

whose role will be outlined in the next section. For now, we consider the initial interaction between N and V in ballot casting. The extensive form of this interaction is shown in Figure 5.1.[4] Nature's first two moves are randomly drawn with equal probability from the action sets $A_{N_1} = \{\text{Top: XY, Top: YX}\}$ and $A_{N_2} = \{\text{Bottom: XY, Bottom: YX}\}$ and will define the layout of the ballot given to the voter.

Upon observing N's moves, V chooses a position to mark, left or right, from action set $A_{V_1} = \{\text{L, R}\}$. In particular, V will choose an action such that a particular candidate will be voted for. V then chooses to keep the top sheet or bottom sheet: $A_{V_2} = \{\text{T, B}\}$. This decision does not influence, of course, which candidate was voted for, however the three previous moves all influence the outcome. This will become important in section 5.4.2.

The privacy of the receipt comes from the fact that this model contains hidden information. Depending on how V moves, either A_{N_1} or A_{N_2} will be hidden from any observer of the receipt. If one were to only observe V's receipt and not both moves by N, they could only determine the outcome to be in a set of outcomes joined with a dotted line in Figure 5.1 but not know which outcome. For all outcome sets, the state of the world could be a vote for Alice or Bob with equal probability. For this reason, the privacy of a Punchscan receipt appears very strong, however this does not imply coercion-resistance.

5.3 Contract-based Attacks

Despite appearances to the contrary, Punchscan receipts can be exploited to bias a voter's choice. This is accomplished through a contract, which is presented to V by the adversary. A contract specifies, for each possible receipt a voter can construct, a payoff the voter will receive for that receipt. Assuming V is utility-maximizing, V will construct her receipt in a way that maximizes her payoff. Using this property, the adversary seeks to offer a contract that will result, on balance, in more votes for his preferred candidate than the other candidates. We study three proposed contracts that accomplish this for two-candidate races, named for their authors: MN [MN07], BMR [BMR07], and KRMC [KRMC10]. These three contracts are not central to their respective works, and thus certain subtleties are glossed over by the authors which we will fill in.

An alternative to contracts suggested in the same literature are scratch-off cards. A scratch-off card, in a race between Alice and Bob, would be a 2×2 matrix, with the rows marked X and Y, the columns L and R, and each cell would contain a random T or B underneath a scratch-off layer. The voter is given a new card and instructed to vote for Alice, scratch off the cell that corresponds to the letter beside Alice's name and the position

[4] An extensive form diagram is a tree, with the root node defined as the first player to move and a vertex defined for each action the player can take for this move.

where this letter appears on the ballot received by the voter. The voter then retains the top or bottom layer, as revealed. Both the receipt and the card must be returned to the adversary, who checks that they are consistent. If the voter does not vote for Alice, the voter must scratch off a cell that does not correspond to either Alice's symbol or the position of the asserted symbol on the bottom layer of the ballot. In both cases, the voter will be caught with probability 0.5—if the scratch reveals T in the former case or B in the latter.

Scratch-off cards are attractive since they fix the adversary's ability to gain votes for Alice, while we will show in Section 5.4.1 that contracts perform worse as the number of candidates increases. By contrast, contracts are attractive because they are informational and can be memorized by voters (especially in the case of voting buying, where the voter has such an incentive). This eliminates the risk of being caught using a scratch-off card or even giving the voter incriminating evidence of the undue influence. In addition, contracts do not need to be secure against physical tampering (scratch-off surfaces can be removed and reapplied). Finally, contracts do not necessarily require the voter to rendezvous with the adversary after the attack. The voter can simply report their serial number and the adversary can retrieve the information from the public record (this assumes the voter does not collude with other voters to misreport their serial number as the serial number of another voter's receipt that coincidently meets the conditions of the contract; a difficult task for the voters to arrange). Our purpose is not to argue that contracts are better than scratch-off cards, merely that contracts have enough interesting advantages to warrant their own thorough study.

5.3.1 Voter Coercion and Vote-Buying

It is useful to distinguish between voter coercion and vote-buying. As mentioned, the contract will offer payoffs in the form of utility. These utilities are in either two or three amounts with strict ordering: $\{u_0, u_1, u_2 \mid u_2 > u_1 > u_0\}$. Generally, a vote-buying contract will promise positive utilities, such as $u_0 = \$0$, $u_1 = \$5$, and $u_2 = \$10$, while a coercive contract will threaten negative utilities, such as u_0 as arson against a home, u_1 as slashed tires, and u_2 as nothing happening. A further distinction is that participation in vote-buying is voluntary, while coercion is involuntary as no rational voters would opt into a negative utility. For example, if each candidate was buying votes, we might expect voters to go out of their way to be paid to vote how they were going to vote anyways. Conversely, if each candidate was using coercion, voters would gain to benefit from purposely positioning themselves to be coerced. We use the term vote-buying to refer to a voluntary contract (with positive utilities) and coercion to refer to an involuntary contract (with at least one negative utility). Contracts could, of course, include both positive and negative utilities.

5.3.2 The MN Contract

The first contract we consider is due to Moran and Naor [MN07]. It is presented by the authors as a vote-buying contract and is w.l.o.g. biased toward Alice.[5] It is as follows:

$$\text{Contract}_{MN} = \begin{cases} u_1 & = \pi_V(L) \\ u_1 & = \pi_V(R, T \quad |\{XY, \_\_\}) \\ u_1 & = \pi_V(R, B \quad |\{\_\_, XY\}) \\ u_0 & \text{otherwise} \end{cases}$$

In our notation, this means that V is given a payoff (π_V) equal to u_1 for any receipt where the left position is marked, or a top sheet with symbols XY and the right position marked, or a bottom sheet with symbol order XY and the right position marked. Any other receipt is given u_0. The underscores denote information that is hidden due to the choice of T or B.

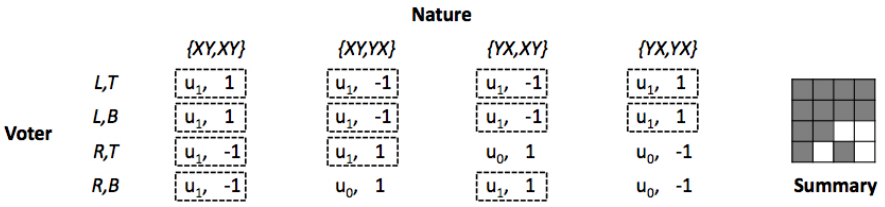
The normal form of the contract is shown in Figure 5.2(a).[6] Since V, the row player, only moves after observing the move made by N, we consider V's best response to each of N's actions separately, which is the highest payoff to V (the first number in the pair of payoffs) in each column. This assumes the voter is utility-maximizing and is only interested in the highest payoff, a simplifying assumption that we will reconsider in Section 5.4.3.

The second payoff in the pair, with a slight abuse of notation, is to the influencer I and not to the column player N. I receives +1 when V votes for Alice and -1 when she votes for Bob. Since I's payoffs are not a function of how much money he is paying to V, this implicitly assumes that money is no object. This is a simplification that we will rectify in Section 5.4.4.

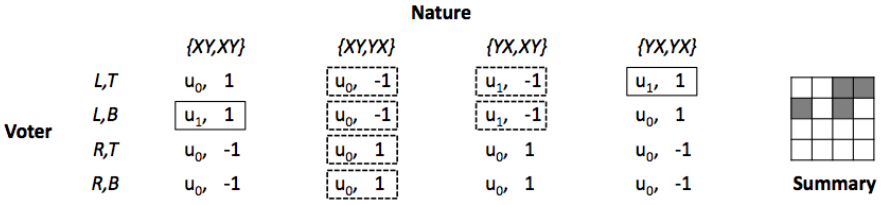
Recall that N chooses each column with equal probability: 0.25. If the first column is selected, the voter will receive u_1 in any case. It is difficult to interpret what these weakly dominant responses mean to a utility-maximizing voter but let us assume the voter will choose randomly between them. This is more problematic in the second column, where the voter has three options: two of which result in a vote for Bob and one for Alice. We could assume the voter, caring only for the payoff, (i) chooses randomly between the two

[5] All the contracts considered in this chapter will be presented in their pro-Alice form for consistency and easy comparison. Due to the symmetric nature of the ballot casting process, any contract can be adapted for Bob instead.

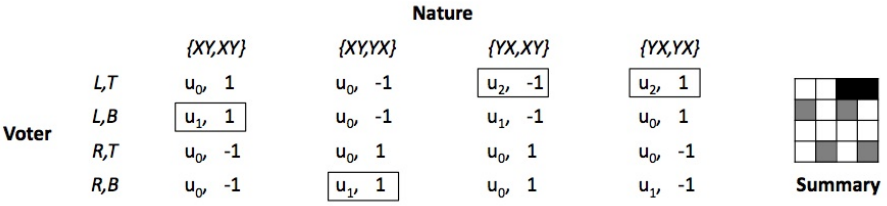
[6] The normal form of a game is a matrix with player 1's action set as the rows and player 2's action set as the columns. The elements contain a tuple: the payoff to player 1 and player 2 respectively for the selection of these actions (however note the deviation from convention in this case). A player's dominant strategy, if one exists, is the selection of an action that will always yield a higher payoff than any other action, and a weakly dominant strategy is the selection of an action that will yield at least as high of a payoff as any other action.



(a) MN



(b) BMR



(c) KRMC

Figure 5.2: Three pro-Alice contracts in normal form. Lined boxes are dominant best responses, while dotted boxes are weakly dominant best responses. The underlying game is sequential with the column player moving first; thus each column is a subgame. *Notational abuse:* the first element of the payoff is to the row player, V, while the second element is to the influencer I; not the column player N. This latter utility distinguishes votes for Alice (+1) and for Bob (-1). The summary captures the payoff to the row player, visualizing higher utilities as darker squares.

candidates or (ii) chooses randomly between the three options. This has an effect on I's expected payoff. The third column is much like the second, while the final column is the interesting one: both options produce a vote for Alice. Thus I is guaranteed a vote for Alice whenever this column is chosen by N.

We can calculate the probability of this contract resulting in a vote for Alice to be 0.625 under interpretation (i) and 0.54 under interpretation (ii). For all outcomes, I will incur u_1, thus the purchase of a full vote for Alice requires manipulating 1.6 voters for a cost of $(1.6)u_1$ per vote under (i) and 1.85 voters under (ii) for a per vote cost of $(1.85)u_1$. Although proposed as a vote-buying contract, it also works for coercion.

5.3.3 The BMR Contract

The second contract is due to Bohli, Müller-Quade, and Röhrich[BMR07], and is presented by the authors as a vote-buying contract:

$$\text{Contract}_{BMR} = \begin{cases} u_1 & = \pi_V(L, T \quad |\{YX, \_\_\}) \\ u_1 & = \pi_V(L, B \quad |\{\_\_, XY\}) \\ u_0 & \text{otherwise} \end{cases}$$

The normal form of the contract is shown in Figure 5.2(b). A curiosity here is the second column, which yields u_0 regardless. If used coercively, with probability 0.25, the voter cannot escape punishment: there is no way, given a ballot like this from N, to please I. For this reason, we consider BMR a sub-optimal approach to coercion (if otherwise equal, we would prefer to eliminate the ambiguity). The probability of the contract resulting in a vote for Alice is 0.625. This outcome will cost I $(0.75)u_1$ (assuming u_0 is zero). The purchase of a full vote for Alice requires manipulating 1.6 voters for a cost of $(1.2)u_1$ per vote. Thus this contract is better than the MN contract for vote-buying.

5.3.4 The KRMC Contract

The final contract is due to Kelsey, Regenscheid, Moran, and Chaum [KRMC10], and is presented by the authors as a vote-buying contract:

$$\text{Contract}_{KRMC} = \begin{cases} u_2 & = \pi_V(L, T \quad |\{YX, \_\_\}) \\ u_1 & = \pi_V(L, B \quad |\{\_\_, XY\}) \\ u_1 & = \pi_V(R, B \quad |\{\_\_, YX\}) \\ u_0 & \text{otherwise} \end{cases}$$

The normal form of the contract is shown in Figure 5.2(c). The contract is similar to BMR, only it uses graduated payoffs and includes an additional clause to resolve the ambiguity in the second column of BMR. Every column contains a strongly dominant response, leaving no ambiguity to a utility-maximizing voter. It can be made to work for coercion (if u_0 is negative, u_1 is neutral, and u_2 is moderately positive), and it works perfectly with vote-buying. The probability of the contract resulting in a vote for Alice is 0.75. This outcome will cost the influencer $(0.5)(u_1 + u_2)$. The purchase of a full vote for Alice requires manipulating 1.3 voters for a cost of $(0.5)(u_1 + u_2)$. Since u_2 needs to be only epsilon greater than u_1, this contract is more effective than BMR and MN; more applicable than BMR; and has less ambiguity than BMR and MN.

Contract Clause		MN	BMR	KMRC
L,T | {XY,__}		u_1	u_0	u_0
L,T | {YX,__}		u_1	u_1	u_2
R,T | {XY,__}		u_1	u_0	u_0
R,T | {YX,__}		u_0	u_0	u_0
L,B | {__,XY}		u_1	u_1	u_1
L,B | {__,YX}		u_1	u_0	u_0
R,B | {__,XY}		u_1	u_0	u_0
R,B | {__,YX}		u_0	u_0	u_1
Perfect: 				

Figure 5.3: A summary of the three two-candidate contracts, delimitated by possible clauses.

5.3.5 The Optimal Contract

We have seen three contracts with different properties and expected votes for Alice. KRMC is the best contract, and in this section, we seek to determine if a contract could do better. Consider Figure 5.3. The leftmost column shows every possible (most specified) clause that could appear in a two-candidate contract. While clauses do not have to be fully specified in each variable, such as the first clause in MN, such general clauses are some combination of the most specified clauses: the combination of the first, second, fifth, and sixth clauses in this case.

For each clause, the second column of the figure contains a small grid. This grid is intended to be a visualization of the payoff matrix, like the summaries in Figure 5.2. Let C be the number of candidates. The rows of the grid represent the voter's binary choice between the top or bottom layer as well as the C-way choice of which position to mark; hence, $2C$ rows. The columns represent the order of the symbols on the ballots. These orderings are random rotations, not full permutations which simplifies the tallying process of Punchscan. There are C^2 possible orderings, not $C!^2$, and hence C^2 columns. Black cells represent the positions in the payoff matrix that will be affected by adding the clause.

For example, if a contract offers a payoff of u_1 for receipts matching the first clause in the figure, u_1 will be added to the two indicated cells in the payoff matrix for the contract: cells (1,1) and (1,2). MN in Figure 5.2(a) is an example of contract that includes such a clause.

The next three columns summarize the three contracts in the literature and the payoffs they award for each clause. This information can be combined into a concise visualization of the contract by layering the grids associated with each clause on top of each other, where the darker squares represent a higher payoff to V for that outcome. The concise form is shown in the bottom row of each contract.

As a reminder of which outcomes result in a vote for Alice, these outcomes are marked with black cells in the perfect contract in the bottom-left of the figure (*i.e.,* the elements in Figure 5.2 with payoffs of 1 to I). We refer to these cells as the Alice region of the grid (and the inverse set of cells as the Bob region). The perfect contract is not possible to achieve with the available clauses; however an optimal contract will resemble it as closely as possible.

Contract Properties. Continue to consider a contract as a grid, with rows $0 \leq j \leq 2C-1$ and columns $0 \leq k \leq C^2-1$. Each element contains u_i with $i \geq 0$. The following three properties exist for any $C \geq 2$:

P1: For each clause with utility u_i in the contract, a column $\hat{k}$ has u_i added to it in the Alice region.
P2: In P1, u_i is always added to the region of each additional candidate in the same row and some column other than $\hat{k}$.
P3: In P2, the (set of) column(s) is either $\{k|\lfloor\frac{k}{C}\rfloor = \lfloor\frac{\hat{k}}{C}\rfloor\}$ or $\{k|k \equiv \hat{k} \mod C\}$. We call these type-A and type-B clauses, respectively.

Most specified clauses include a top or bottom layer, T or B, and a marked position that we will now call P_m, where $0 \leq m \leq C-1$, instead of the two-candidate specific terms left and right. Clauses also include an ordering of symbols to appear on the receipt. Consider an arbitrary ordering to be the canonical ordering $\hat{o}$. The other possible orderings are generated by rotating this ordering right or left, which we denote with functions `ror`() or `rol`(). For example, if $\hat{o} = XY$ then $\texttt{ror}(\hat{o}) = YX$. This set has closure, such that $\texttt{ror}^C(\hat{o}) = \hat{o}$ (*i.e.,* `ror` applied C times to an ordering is the same ordering). There are $2C^2$ possible (most specified) clauses.

Defining Optimality. We say a contract is optimal with respect to Alice if it has the following three properties:

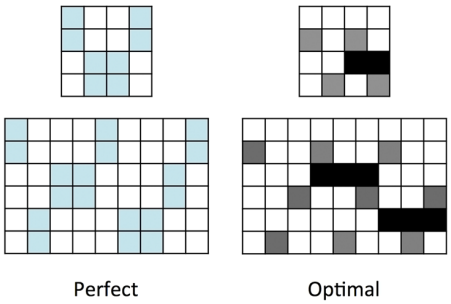


Figure 5.4: Summaries of optimal contracts for two (top) and three (bottom) candidates. The left side shows the perfect contract and the right side shows the optimal contract generated by our algorithm.

O1: The contract maximizes the expected probability of a vote for Alice (from utility-maximizing voters).
O2: The contract contains no columns where the highest utility is shared between more than one candidate (unlike the first three columns in MN).
O3: The contract contains no columns with all u_0 (unlike the second column in BMR). This is a special case of **O2**.

We say a contract contains no ambiguity if it meets **O2** and **O3**. Eliminating ambiguity makes it simple to state the expected value of votes for Alice: it does not require assumptions about how voters respond to arbitrary decisions.

Optimal Contract Generation. We construct a simple, $\mathcal{O}(C)$ greedy algorithm to select an optimal contract. The algorithm selects a contract, w.l.o.g., for the first listed candidate (*i.e.,* a pro-Alice contract). There are many contracts satisfying the properties for optimality—this algorithm finds one instance. It is given in Algorithm 1.

Algorithm 1: Optimal Contract Generation

Add to contract: $u_1 = \pi_V(P_0, B|\{\_\_, \hat{o}\})$
for m *from* 1 *to* $C-1$ **do**
  Add to contract: $u_1 = \pi_V(P_m, B|\{\_\_, \texttt{ror}^m(\hat{o})\})$
for m *from* 1 *to* $C-1$ **do**
  Add to contract: $u_2 = \pi_V(P_m, T|\{\texttt{rol}^m(\hat{o}), \_\_\})$
Add to contract: $u_0 =$ otherwise

Figure 5.4 shows the result of the algorithm for two candidates and three candidates. In Appendix A, we prove Algorithm 1 is optimal according to **O1**, **O2**, and **O3**.

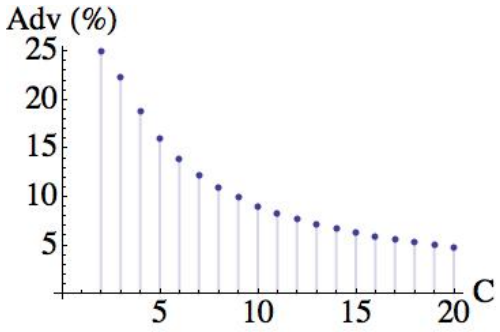


Figure 5.5: The advantage of an optimal contract over a random selection as C, the number of candidates, grows. As seen, the advantage appears asymptotic to 0 in the number of candidates.

KRMC Optimality. Recall that Algorithm 1 finds one instance of the many contracts satisfying the properties for optimality. Running this algorithm for the two-candidate case produces the following contract:

$$\text{Contract}_{opt} = \begin{cases} u_2 & = \pi_V(R, T \quad |\{YX, \_\_\}) \\ u_1 & = \pi_V(L, B \quad |\{\_\_, XY\}) \\ u_1 & = \pi_V(R, B \quad |\{\_\_, YX\}) \\ u_0 & \text{otherwise} \end{cases}$$

This contract is also illustrated in Figure 5.4. It is slightly different than KRMC but is equivalent with respect to **O1**, **O2**, and **O3**. Therefore since the output contract is optimal (as we show in Appendix A), KRMC is as well.

5.4 Extending the Base Model

5.4.1 Multiple-Candidate Contracts

For our first extension to the basic two-candidate contract, we consider the optimality of a contract as the number of candidates is increased from two to an arbitrary number, C, of candidates. **P1**, **P2**, and **P3** still hold. For an arbitrary C, an optimal contract for Alice can be constructed from adding type-B clauses to the first C columns with payoff u_1 in Alice's region, and then type-A clauses in every additional C^{th} column with payoff u_2. The proof in Appendix A covers the C-candidate case and establishes some additional properties of such contracts.

Figure 5.5 shows that the advantage an optimal contract offers over forcing a utility-maximizing voter to vote for a random candidate. The two-candidate case had a probability

of 75% of resulting in a vote for Alice, which is a 25% advantage over a random choice between two candidates. The probability of a vote for Alice in the three-candidate case is 56%, which is only a 22% advantage (over a random choice between three candidates). Likewise, as C increases, the adversary's advantage decreases.

5.4.2 Reordering the Game

We now consider the order of play. If V were to choose either R or L prior to N choosing the ballot layout, the candidate voted for would be random. Enforcing this in a contract, without any additional clauses, could be an effective denial of service attack—but it is no better than simply paying the voter to not vote at all. Thus if V is to vote with intention, she can only choose between R and L after observing the moves by N. However the outcome of the game is invariant to whether V chooses T or B, therefore this move could be safely relocated in the sequence of events. If it was chosen by V prior to observing the moves by N, then no contract can be formed that would favour Alice or another candidate. In other words, this simple change solves the problem.

To see why, consider again the properties in Section 5.3.5. In this new extensive form, **P1** and **P2** still hold. However **P3** does not. If the top sheet is selected, then the set of columns in **P3** will only be $\{k|\lfloor\frac{k}{C}\rfloor = \lfloor\frac{\hat{k}}{C}\rfloor\}$. Likewise, if the bottom sheet is selected, the set of columns will only include $\{k|k \equiv \hat{k} \mod C\}$. With this symmetric pairing of columns, there is no way to asymmetrically win a column for Alice without losing one to another candidate.

Requiring the voter to select the top or bottom layer before seeing the ballot is a known solution, and the Punchscan procedure has been subsequently modified to reflect this change. However this change does cause the privacy of the system to be contingent on poll worker procedure, which is a weak foundation for something as critical as ballot secrecy. From our experience observing elections, we would anticipate that poll workers would not follow procedures exactly. This is especially true when a deviation from the procedure does not interfere with the voters' ability to cast their ballots, and when the poll-workers do not have a solid mental model of why a procedure is important.

In general, removing decisions that a voter must make during the voting process, especially arbitrary choices made after observing the actions selected by nature can help resolve issues of coercion. However one choice can never be eliminated: selecting a candidate to vote for. For some obfuscation mechanisms in E2E systems, this decision alone may be exploitable. Thus, there is no simple trick—vote casting procedures must always be carefully examined.

5.4.3 Voter Types

So far, we have considered V to be utility-maximizing and thus follows the contract fully. However, this is not necessarily the case, especially for vote-buying. There may be some voters who will forgo payment and always vote for Alice or Bob. In this case, they are still utility-maximizing: they receive utility that is external to the contract from their political convictions or expected benefits from an elected candidate. Our use of the term utility-maximizing should be interpreted as maximizing only the utility internal to the contract. There may be other "vengeful" voters who would punish the adversary whenever possible: for example, by always choosing to vote contrary to the adversary when given the choice. In the next section, we also consider opportunistic voters who will sell their vote if they are already intending to vote for the adversary's candidate. In all of these cases, the true type of the voter is hidden from the influencer.

Consider a simple split between the fraction of utility-maximizing voters (α) and vengeful voters ($1-\alpha$) in the coercion model. The adversary will accept any payoff in the set of best responses. In MN, the expected votes for Alice from a vengeful voter is 0.25. Thus to make ground for Alice, the following expression should hold: $(0.54)\alpha + (0.25)(1-\alpha) > 0.5$. The means $\alpha > 0.86$ or at least 86% of the voters need to be utility-maximizing for the attack to work. Using the same analysis for BMR, recall that in BMR it is possible for V to obtain a layout from N that has u_0 as a payoff for all moves by V. As a result, a vengeful V can always vote for Bob, even if the payoff is u_0 since V could have plausibly received a bad ballot type. As a result, $\alpha > 0.8$ which means BMR can tolerate a higher proportion of vengeful voters than MN while maintaining profitability. For KRMC, a vengeful voter can at best vote for Bob on half of the columns (by receiving u_1 on the fourth column). Thus any $\alpha > 0$ will produce profitability for KRMC, making it the most resilient of the three.

5.4.4 Money is an Object

Consider the vote-buying model. In this case, voters could simply choose to reject the contract, receive u_0, and vote for either Alice or Bob. However, occasionally such strategies will coincidentally allow them to meet the terms of the contract and be paid. Say that the fraction of voters rejecting the contract and voting for Alice is p_a and for Bob is p_b. The fraction that are utility-maximizing and opt into the contract is, as before, α, and the vengeful voters make up the remainder. Unlike in the coercive case, vengeful voters will accept u_0 and thus act like voters in p_b. For KRMC, the expected amount of money paid by the adversary to a voter of a hidden type is,

$$0.5(u_1)(p_a) + 0.25(u_2)(p_a) + 0.25(u_1)(p_b) + 0.25(u_2)(p_b) + 0.5(u_1)(\alpha)$$
$$+0.5(u_2)(\alpha) + 0.25(u_1)(1 - p_a - p_b - \alpha) + 0.25(u_2)(1 - p_a - p_b - \alpha)$$

Before we assumed that money was no object, so the value of this expression is irrelevant. However if money does matter, then the adversary must ensure that he is not paying more for a vote than it is worth to him. The difference between a voter in p_a and α can be rephrased: the latter place less value on their vote and thus will choose to accept a payoff that is higher than the amount of value they place on their vote. Let U_v be the value of u_1 such that α voters will accept the contract; in other words, the maximum value a voter in α places on their vote. Since u_2 only needs to be marginally greater than u_1 (*i.e.,* $u_2 = u_1 + \epsilon$), we can assume for simplicity that they are equivalent.[7] Furthermore, assume that p_a is the same as p_b, since close elections will more plausibly have attempts at undue influence. This reduces the equation above to

$$U_v(0.5 + 0.25(p_a) + 0.5(\alpha)).$$

For it to be profitable for the vote buyer, he expects to influence a share of $0.75(\alpha)$ votes in favour of Alice, and if a vote is worth on average U_b to the buyer then

$$U_v(0.5 + 0.25(p_a) + 0.5(\alpha)) \quad < \quad U_b(0.75(\alpha)).$$

This expression forms a ratio between how much a vote is worth to the coercer and how much it is worth to the voter, and how large the ratio must be for KRMC to be profitable. For example, if $p_a = p_b = 0.45$ and $\alpha = 0.10$, the ratio is 8.83. This means that if 10% of voters value their vote at less than, say, \$10, the buyer should only exploit this opportunity if a vote gained is worth at least \$89 to him. For three candidates, $p_a = p_b = p_c = 0.30$ and $\alpha = 0.10$, the vote should be worth at least \$96 to him.

5.4.5 Coercion Contracts in other E2E Systems

Ballot spoiling. One line of manipulation attack can exist in systems that are not diligent in spoiling ballots. If an attacker has a line of communication with the voter, the

[7] This is a consequence of the specific voter types we model. A type that is plausible, but we do not consider, is the voter that has a preferred candidate but is willing to switch for a certain amount of utility. This would require a more sophisticated model where the difference between u_1 and u_2 would be relevant. It would also be harder to analyze and require a distribution on the values for which voters would be willing to forgo their preference.

voter can be instructed to mark her ballot and wait for further instruction. The attacker then communicates to the voter to either spoil the ballot or cast it. If the spoiled ballot is not protected or destroyed, the attacker may consult it to see how the voter would have voted had the attacker instructed the voter to cast the ballot.

The line of communication can be eliminated by using random material on the ballots to determine the instruction. For example, if spoiled ballots were published in Scantegrity, the adversary could instruct the voter to vote for Alice but only cast the ballot if the last letter of the confirmation code is between A and P. Otherwise, she spoils it. The voter's confirmation code should always be between A and P, but every ballot spoiled but this voter must have a confirmation code from Q to Z beside Alice—something the voter cannot arrange if she votes for a candidate other than Alice. In Chapter 4, we avoided this line of attack by having spoiled ballots shredded in front of the voter, without the poll worker seeing the contents of the ballot.

Benaloh observes that the same attack applies in his voter initiated auditing [Ben06, Ben07], and systems based on it [SDW08, Adi08], if the adversary can observe how many ballots the voter spoils and audits before choosing one to cast. For example, the adversary could instruct the voter to construct a ballot for Alice but only cast the ballot if the ciphertext (in binary) is returned with the bits 00. For each other ciphertext, the voter will audit it and retain a receipt of who the voter constructed the ballot for.

ThreeBallot. Henry, Stinson, and Sui show how coercion contracts can be used in ThreeBallot [RS07] when the information on the bulletin board is included in the coercer's strategy [HSS09]. By specifying a voter casts her ballot with a certain pattern that is likely to be unique in an election with long (in terms of candidates and/or contests) ballots, the coercer can verify the voter's compliance with a certain probability.

A different illustration of how important the order of ballot casting is in E2E systems can also be observed in ThreeBallot. Although not a coercion contract *per se*, Clark, Essex, and Adams point out that ThreeBallot receipts leak partial information about how the voter voted if the voter fills out her ballot before choosing which part to retain as a receipt [CEA07]. Later, de Marneffe, Pereira, and Quisquater show that this bias can be eliminated by reordering the voter's actions [MPQ07]. In the reordered game, the voter first fills in a single vote for each candidate (recall Figure 3.1(c) from Chapter 3), then randomly chooses her receipt, and then votes for her preferred candidate on a ballot other than the one she chose as her receipt.

5.5 Concluding Remarks

We have shown how game theoretic-models can be applied to analysing coercion contracts in E2E election systems. We developed an algorithm for devising optimal contracts, proved that KRMC is optimal in the two candidate case, and found that the effectiveness of contracts decreases with the number of candidates. We also showed that no more than three levels of utility are needed and that contracts are costly for the adversary.

We hope our study of contracts in Punchscan, and the tools we have used in our analysis, is of assistance to the designers of E2E systems. The more we understand these attacks, the easier it will be to design against them.

Future work. We conclude with a few avenues for future work. In paper-based elections, where unrecoverable errors are possible, voters are typically given the option to spoil a ballot and receive a new one. Future work could examine the impact of spoiling on coercion contracts in realistic scenarios, like being allowed up to two spoiled ballots: some voters will spoil to try and receive higher payoffs, others may spoil to avoid meeting the adversary's demands. Voters must strategize whether spoiling is likely to increase or decrease their fortunes when the payoffs are ternary or when there are multiple contests on the ballot, each with its own payoff. It may also be plausible for the adversary to observe when the voter spoils a ballot, and he may adjust his own strategies accordingly.

Our definition of optimality assumes voters are utility-maximizing, and we later study the performance of these contracts in a setting for which they were not optimized: voters with hidden types. We conjecture that reoptimizing the contracts for this setting would not change the contract; however, we leave proof of this for future work. A final topic for further exploration is the potential for adversaries to employ screening techniques to differentiate between voters with hidden types. For example, the payoff could include a contribution to one candidate's campaign to prevent supporters of another candidate from accepting the contract if their receipt coincidentally meets its conditions.

Chapter 6

Implementing Random Beacons with Financial Data

This chapter is adapted from published work supervised by Urs Hengartner [CH10].

6.1 Introductory Remarks

In many countries, including the United States, electronic elections have become predominant. Some electronic voting technologies additionally provide a paper-based record of each vote—*e.g.,* optical scan and DRE machines with voter-verified paper audit trails. This paper record can be compared with the electronic tally, and while both tallies could be consistently manipulated, this check does provide some level of assurance that the electronic tally is correct. It is not feasible to perform this check for all precincts; however by randomly selecting a small number of precincts in an unpredictable way, excellent statistical assurance of the consistency of these tallies can be achieved.

This mathematical approach to protecting against errors and manipulation in elections can be taken a step further. Cryptographic election systems use mathematical techniques throughout the entire voting process to provide a stronger notion of integrity: even if all the records of all the ballots are subverted, the manipulation will be reliably detected. In many cryptographic systems, assurance is established by challenging the system to prove computations were performed correctly. As with choosing precincts, challenges in cryptographic voting must be random.

This chapter is concerned with choosing a source of randomness to generate these random selections or challenges. We will speak abstractly and phrase the nature of the random action as selecting *units* to *audit* (following [Res09]). It is hopefully intuitive that

if the source of randomness used to select units is predictable or can be influenced, the statistical assurance loses strength.

One proposal for an unpredictable source of randomness is financial data (we review a number of other proposed methods in the next section). While not broadly adopted, this technique has been used in at least two binding cryptographic elections: a university election with Punchscan [ECCP07a] and a municipal election with Scantegrity II [CCC+08]. In both cases, cryptographic tools were used to transform the random financial data into a useful form. This may lead some to object to using this type of protocol in non-cryptographic voting contexts, like precinct selection. Our position here is neutral—its application to cryptographic voting is motivation enough for this study, and we note that if cryptographic techniques were permissible for use, then this approach does provide a more publicly verifiable challenge than, say, rolling dice in a room.

It may be acceptable to the reader that financial data (*e.g.,* stock market prices) exhibits some unpredictable behaviour, but it is not likely intuitive how much randomness there is. In this chapter, we combine a model from computational finance with techniques from information theory to estimate the amount of entropy in the daily closing prices of a number of common stocks (the same set used in the Scantegrity II election). In our sample, a typical closing price has an estimated 6–9 bits of entropy per trading day. When we consider the joint entropy between two highly correlated stocks, we find that the joint information is less than a single bit, suggesting that using a portfolio of stocks is a good method of increasing the pool of entropy.

Finally, we present a straight-forward protocol that can be used to take the prices in their raw form and produce a near-uniform random bit string. Our intention is that some entity will publish the output from our protocol on each day that the market closes as a service. As an incentive to offering this service, we allow the entity to mix in their own randomness. This is done in a way that is transparent and verifiable to everyone to ensure that the entity will be caught if it misbehaves. The output can then be used either directly to select units for auditing, or if many bits are required, the output can be used to seed a pseudorandom generator. The output is may also be useful for non-voting cryptographic protocols where a beacon or common reference string is required.

Contributions. The contributions of this chapter can be summarized as:

- a general technique for estimating the amount of entropy in a stock market closing price that is extensible to any simulatable model in computational finance,
- a set of empirical estimates of the entropy in a representative portfolio of stocks, under the conservative assumption that the Black-Scholes model holds for market behaviour,

- a demonstration of the impact on entropy that the four parameters—drift, diffusion, initial price, and time period—have within the range of our data,
- an estimate of the mutual information shared by correlated stocks within the same business sector, and
- a protocol for regularly publishing verifiable random bitstrings based on stock prices.

6.2 Related Work

6.2.1 Public Randomness in Cryptography

Many cryptographic protocols require randomness. For most applications, the randomness is only for private use. However, in some scenarios, publicly verifiable randomness can be useful. Two examples are fair exchange and interactive proofs.

In multiparty protocols, it is sometimes advantageous for a malicious party to abort early after they have received information from the other parties and prior to sending their own. An early proposal to mitigate this threat in the case of contract signing, due to Rabin, introduced the notion of a *beacon*, which is an unpredictable and random value published at regular intervals by a trusted third party [Rab83]. Later, contract signing protocols without beacons were proposed [EGL85].

In interactive proofs, including those with the zero knowledge property, a verifier interacts with a prover to become convinced of some fact. A common form of an interactive proof follows three steps (sometimes called a Σ-protocol): the prover commits to some information, the verifier generates a random challenge, and the prover, bound to the information they committed to, must respond convincingly to the challenge. In the earliest model, due to Babai, the verifier was bound to only producing random bits visible to both the prover and verifier— *public coins* [Bab85]. Goldwasser and Sipser later proved that this class is, for all practical purposes, equivalent to a class with coins that are visible to only the verifier: *private coins* [GS86]. A party independent of the verifier can also become convinced of the proof if they believe the coins were random and unpredictable to the prover. In protocols where the verifier's role is only to choose a random challenge, Fiat and Shamir showed that the verifier can sometimes be eliminated entirely by hashing a transcript of the protocol up to that point and using the output as the challenge, making the proof non-interactive [FS86]. More recently, Groth and Sahai (among others) demonstrated non-interactive proof systems without random challenges at all, based on bilinear pairings [GS08].

6.2.2 Public Randomness in Elections

Public randomness is used in post-election auditing procedures, usually for the selection of precincts for manual recounts. Cordero *et al.* establish a set of desirable properties for generating the randomness: it should be simple to understand, efficient to execute, difficult to manipulate, and verifiable after the fact [CWD06]. The authors consider a number of possible mechanisms: cryptographic coin tossing protocols, drawing tickets from a hat, lottery machines, random number charts, shuffling cards, flipping physical coins, and rolling dice. They then devise an algorithm for random precinct selection using dice.

Three main criticisms of the dice-based approach are available. Clark *et al.* point out that protocol is only verifiable to those in the room [CEA07], Calandrino *et al.* point out that the number of rolls can become infeasible for reasonably sized districts [CHF08], and both note the difficulty of determining whether the dice are fair.[1] Calandrino *et al.* further suggest a hybrid protocol that involves both dice and random draws in order to generate a seed, which is expanded with a cryptographically secure pseudorandom number generator (CS-PRNG). They allow the procedure to be video recorded to expand verification to those not present.

Of the other mechanisms suggested by Cordero *et al.*, at least two others have been examined further. Hall finds that a particular real-world use of a lottery machine, where numbered ping pong balls were drawn from a hopper, yields non-uniform results and proposes a fix [Hal08]. Rescorla examines the use of random number charts, where a seed is used to index into a random spot in the chart [Res09]. Since the chart is limited in size, the seed is low-entropy (*e.g.,* 16 bits) and the state space is small. This has various consequences, including difficulty in the selection of a pseudorandom stream that is statistically independent of other possible streams. Rescorla finds that using the same seed with a PRNG yields better properties. Clark *et al.* use a different statistical approach but also find that low-entropy seeds expanded with a PRNG can be secure.

6.2.3 Cryptographic Elections

The use of algorithmic or cryptographic techniques, like (CS-)PRNGs, has been criticized for potential use in normal elections as being difficult to understand and computer-reliant [JGM+07]. However in cryptographic elections, where extensive cryptographic techniques are already being used, it is obviously congruent. In cryptographic election systems,

[1] Interestingly, this problem has a solution although it appears to never be mentioned in the voting literature. The solution builds on von Neumann's famous result for generating a fair coin from an unfair coin: flip the unfair coin twice, if it is both heads (HH) or both tails (TT), discard the trial (output $\perp$). If it is heads followed by tails (HT), output a heads (H), and if it is TH, output T. This can generalized to n-sided dice [JJSH00].

interactive proofs and arguments are typically used. For this reason, if one accepts the random oracle assumption, the mentioned Fiat-Shamir heuristic is often the easiest mechanism to implement. Systems like Helios, used in a binding student election [AMPQ09], use this approach [Adi08]. However Fiat-Shamir is suitable only when the challenges are drawn from a large space—a space larger than that which can be exhaustively searched [GK03]. In both Punchscan and Scantegrity, the numbers are used to select half of 10 to 20 units (committed to by the election officials) to audit in a post-election, cut-and-choose argument that proves the tally was not manipulated by the officials [ECCP07a, CCC+08].[2] For this particular audit, the challenge-space is 2^{20} (for 20 units) and an accepting Fiat-Shamir challenge that hides manipulation can be expected with 2^{19} rewinds on average,[3] which is computationally feasible today. This motivates the use of public randomness instead.

6.2.4 Stock Market for Public Randomness

Stock market data has been suggested for use as public randomness in several publications. Waters *et al.* propose a service called a *bastion*, which is similar to Rabin's notion of a beacon, only it produces random cryptographic puzzles instead of random numbers [WJHF04]. These puzzles are issued by online services to clients to solve prior to gaining access, which helps prevent denial of service attacks. The authors relate bastions to beacons and specifically suggest the option of financial market data for implementing a beacon. A subset of these authors, Halderman *et al.*, later propose and implement a more general framework, Combine, for harvesting challenges from various online data sources, that can include financial data, to thwart Sybil attacks [HW07].

Stock market data has also been suggested for randomly selecting an IETF nominating committee, along with lottery numbers and sporting outcomes [Eas06]. However in a later update, financial data was specifically advised to be discontinued because it is not always reported consistently from all sources [Eas04]. We address this issue later in Section 6.3.4.

Finally, stock market data was proposed by Clark *et al.* for use in the Punchscan cryptographic election system [CEA07]. This approach of using stock market data is preserved in the Scantegrity II system, which is related to Punchscan through contributors and codebase. However for Scantegrity II, Rivest implements a novel protocol for converting the

[2]Here, the cut-and-choose argument follows the same three-step procedure as a Σ-protocol: commit, challenge, and response. It is not a zero-knowledge proof for technical reasons that are beyond the scope of this chapter.

[3]By rewind, we mean that the malicious election authority would guess what the challenge will be, hide their manipulations such that the guessed challenge will not uncover them, use Fiat-Shamir to generate the actual challenge, and check if it matches the guessed challenge. If it does not, then they can rewind and guess a new challenge. This approach assumes that the input to the hash can be manipulated in 2^{19} different ways to generate unique challenges on each rewind. In Punchscan and Scantegrity, it can be.

portfolio of closing prices into pseudorandom bits.[4] Two binding elections were conducted using stock market data for public randomness: a student election in Canada in 2007 with Punchscan and a municipal election in Maryland in 2009 with Scantegrity II.

6.3 Model and Assumptions

Our primary interest in this chapter is the use of financial data as a source of entropy for creating random and unpredictable challenges. If they are truly unpredictable, these challenges can be used in cryptographic voting protocols, particularly zero-knowledge and cut-and-choose protocols, to eliminate the need of a verifier to generate the challenges. While this approach could be used anywhere a challenge is needed, it is especially relevant when the efficient Fiat-Shamir heuristic cannot be used. We are motivated to examine stock prices because of their actual use in binding elections.

Note that random and unpredictable mean subtly different things. If an adversary is able to *set* a challenge to a known value, it is not unpredictable to her. However the value may be statistically random and have high entropy in that sense. Our approach is careful to model the uncertainty of the adversary (or anyone) in predicting the outcome of a stock price. This is different from directly computing the statistical randomness contained in financial data, which could lead to a wrong estimate of the *adversarial* uncertainty.

In this section, we introduce the model that we will use to simulate the movement of stock prices. We assume the reader has no background in computational finance. This model will be used to estimate of the amount of uncertainty in a stock price in the next section. Here, we also address some other potential concerns with using stock prices; namely, manipulation and consistent reporting.

6.3.1 Terminology

Let S_{t_i} be the price of a stock at some time t_i. For time-period T, let S_0 and S_T represent the stock's initial and final price, respectively. Define r to be the risk-free interest rate: the interest-rate on a safe asset, like a government bond, with value β_{t_i}. If r is continuously compounded over period T, a bond initially worth β_0 becomes worth $\beta_T = \beta_0 e^{rT}$.

A *Wiener process*, W_t, is a continuous time stochastic process with the following properties:

[4] R. Rivest, 2009. See: `get_latest_djia_stock_prices.py`, `pre_election_audit.py`, and `post_election_audit.py`. `https://scantegrity.org/svn/data/takoma-nov3-2009/PUBLIC/PUBLIC/`.

- $W_0 = 0$.
- $W_t \sim N(0, t)$, where $N(0, t)$ is a normal distribution with mean 0 and variance t.
- For all intervals in time, $t_b - t_a$, $\Delta W = W_b - W_a$ is independent from all other (non-overlapping) intervals.

A Wiener process is also known as standard Brownian motion, and since future values of the process depend only on the current value, it is an example of a Markov process. *Geometric Brownian Motion (GBM)* for random variable S_t adds a linear constant, μ, to the process, which is known as drift, and also scales the variance of the Wiener process, W_t by the constant σ, known as diffusion:

$$\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sigma X_t \mathrm{d}W_t$$

When S_t represents the value of some instrument, μ is termed the growth rate or expected rate of return. When this is greater than the risk-free rate, it is termed an excess return. The diffusion, σ, is termed the volatility. W_t generates the random fluctuations in price and σ scales them. When volatility is estimated based on past performance of the stock, it is termed historic volatility.

6.3.2 Black-Scholes Model

The *Black-Scholes model*, or Black-Merton-Scholes, is used to model financial markets and determine the value of derivatives (a financial instrument whose value is dependent on the value of an underlying asset) [BS73, Mer73]. We only use the model to study the movement of the underlying asset; in this case, the assets are stocks. The model is based on the following assumptions.

1. There are no transaction costs or dividends.
2. Over time, the asset price is a real number: $S_t \in \mathbb{R}$.
3. Over time, the asset price follows a GBM (as defined above):
   $\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t$.
4. Over time, μ and σ are constant valued.
5. There are no arbitrage opportunities.[5]

[5]Through an argument omitted here (see [Sey09]), this effectively means μ is modelled with r (the risk-free rate) when pricing options. Since we are not pricing options, we use historic volatility to estimate values of μ.

Nearly every mathematical model of a financial market has its criticisms. Black-Scholes is very well-known[6] but has also been controversial, primarily for being too tame of a representation for markets like stocks or commodities. For pricing derivatives, underestimating the volatility of the market can lead to catastrophic loss and thus using models with higher volatility, like Jump-Diffusion models, are a more conservative approach (however they also lead to less competitive pricing) [Sey09]. In our case, we are using stock prices to generate random challenges. It should be intuitive that the entropy in a stock price will increase with higher market volatility (if not, we show this in section 6.4.4), and so the conservative approach for our purposes is the exact opposite. We use Black-Scholes model because, if anything, it errs on the side of not having enough volatility and therefore will be useful in determining a plausible lower-bound on entropy.[7]

6.3.3 Market Manipulation

Since trades are the mechanism that moves the price of a stock in the real world, the closing price of a stock could be manipulated through unnatural sales or purchases, particularly near the closing time of the market. This manipulation is theoretically possible and has been performed on exchanges in developing markets; however there is broad agreement that it is difficult to perform on stocks one might find on an established exchange like the NYSE or NASDAQ. We also note that it is illegal in all major exchange markets.

If manipulation were possible, it could be used in the context of cryptographic voting for the following attack: prior to committing to the election data, an adversary creates a guess for the closing price of each stock that will be used. The adversary then determines what the challenge would be if these guesses turn out to be correct, and hides any electoral fraud in the units that will not audited under this envisioned challenge. The manipulated data is then committed to. Later, the adversary buys shares to raise the price of any stock that is below the guessed value and sells shares (short sells, if the adversary does not hold the shares) to lower the price of stocks over its targeted price. If all the prices close exactly on target, the fraud will escape detection.

There are a large number of practical issues with such an attack (its detectability and illegality, the high volatility of prices near closing time, the use of matching algorithms in determining a closing price, regulation concerning short selling after downticks, etc.) but it is theoretically possible. Even if volume is factored into the randomness, the adversary could choose an unusually high target volume to avoid overshooting it while manipulating.

[6] Merton and Scholes received a Nobel prize for developing it. Black unfortunately did not live long enough to join them.

[7] If, alternatively, the stock market is more predictable than Black-Scholes, our estimates will be wrong. We do not consider this at length as it is not an advocated position, even by a minority, within the financial community, nor is it supported by the empirical evidence.

Market manipulation is considered for different reasons by the financial community. One type of financial derivative that can be purchased is a *barrier option*, which operates like a regular stock option (vanilla option) with the added condition that if the underlying asset goes above (or below) a predetermined price (the barrier) at some time interval (such as the daily closing price), the option becomes worthless. If market manipulations were feasible, they could be used to bump stocks over (or under) a barrier.

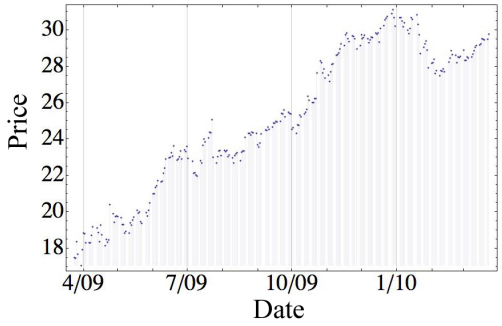
There is broad agreement that this type of manipulation is difficult if the market is volatile and liquid, and/or if the barrier event must happen multiple times (so-called Parisian options [CJPY97]). An empiric study of manipulations in the NYSE, NASDAQ, and other markets during the period of 1990–2001 confirms that manipulations of this type are rare and confined to illiquid stocks [AW03]. Despite the theoretical possibility of manipulation, barrier options continue to be written/held by banks and investors [Sch03]. We also note that these manipulations have a relatively crude goal: to move the stock up or down. In the case before us, manipulations would have be highly calibrated to result in a stock landing on an exact price (to the nearest cent). For the reasons outlined in this section, we consider manipulation infeasible with the selection of liquid stocks on an established exchange. The election we are studying used the 30 stocks in the Dow Jones industrial average, which meets our criteria.

6.3.4 Official Closing Price

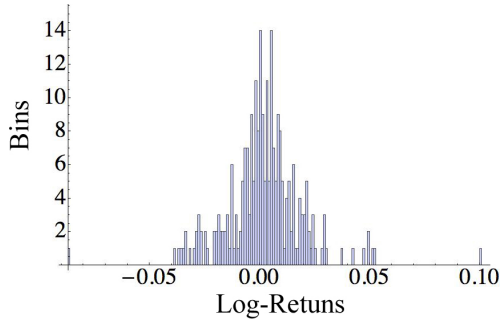
A practical requirement is that closing prices are reported consistently across publications. For example, in the Scantegrity II municipal election, both closing prices and closing volumes (the number of shares traded that day) were used. Auditor Ben Adida reports that the volumes that he accessed differed slightly from those used to generate the challenge, which could be due to differences in rounding, inconsistent reporting, or after market trades.[8] Since the data is being used to generate relatively small challenges, a malicious election authority could slightly perturb the volumes from their actual values until a suitable challenge is generated that hides any fraud. Even though this value differs from the reported values, it is indistinguishable from the scenario where the volumes were changed by the publisher.

For this reason, we recommend that only closing prices are used and not volumes. As the value of options and derivatives depend on the exact closing price of a stock, infrastructure for publishing a uniform closing price is in place. The official closing price is algorithmically determined (*e.g.,* by a closing cross) in a transparent way and then multicast by the Consolidated Tape Association (CTA), typically 15 minutes after the

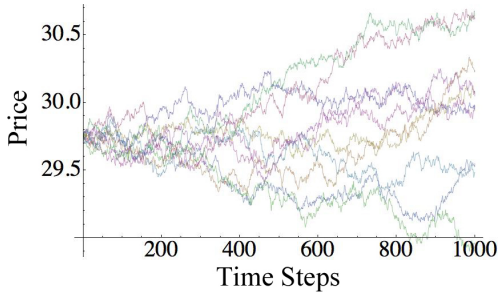
[8] B. Adida. Takoma Park: Meeting 2. `http://benlog.com/articles/2009/11/02/takoma-park-meeting-2/`.



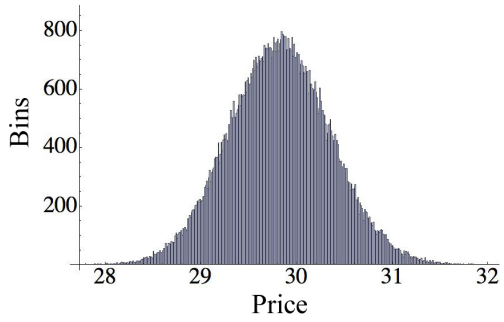
(a) MSFT price history



(b) Histogram of Logarithmic Returns



(c) Monte-Carlo Simulations



(d) Histogram of Final Price

Figure 6.1: Estimating the entropy for MSFT. We (a) import one year of closing prices, (b) measure the relative price changes using logarithmic returns, extract estimates for drift and diffusion parameters and use these to (c) simulate the price over the next day, and (d) compute a histogram of simulated closing prices.

close of the markets. It clearly indicates which trades are considered after-market. Some newspapers or financial data sources may provide a "closing price" that is adjusted by after-market trades—these should not be used. A third party publisher may also make a mistake. We recommend that election officials (or the beacon service provider) check the closing prices from a few sources for consensus before generating the challenge, assuming they do not have direct access to the CTA multicast.

6.4 Entropy Estimates

In this section, we use the Black-Scholes model to estimate the entropy in a closing price. We illustrate the process by using the stock Microsoft (MSFT), which is included in the portfolio of stocks that we are ultimately interested in—the Dow Jones industrial average.

6.4.1 Historical Drift and Diffusion

Figure 6.1(a) shows the closing prices of Microsoft from March 23, 2009 at \$17.95 until March 23, 2010 at \$29.88. Let this series be $\{S_0, S_1, S_2, \ldots S_T\}$. In this case, $T = 251$ given the number of trading days in the provided interval. From this data series, we can calculate the relative price changes using

$$R_i = \ln\left(\frac{S_{i+1}}{S_i}\right),\ 0 \leq i \leq T-1. \tag{6.1}$$

R_i is called the log (or continuously compounded) return. It is positive for relative increases in the stock price and negative for decreases. A histogram of R_i values for Microsoft is shown in Figure 6.1(b) with bin size 0.005. The distribution of R_i for this example is roughly normal, and any deviation from a normal distribution, as we will now derive, is evidence against the Black-Scholes assumption of asset prices following geometric Brownian motion.

With GBM, asset prices are modelled as

$$\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t. \tag{6.2}$$

With a logarithmic transform of S_t and application of Ito's lemma, GBM can be found to have the following analytic solution:[9]

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right). \tag{6.3}$$

Due to the W_t term, this solution is a process. Thus if we are given a set of (S_0, S_t) pairs, we can estimate the values μ and σ to fit this process.

Let Δt be the sampling interval relative to the measured period. For example, if we sample daily prices and want the annualized distribution, Δt would be 251. In our case, we sample daily values to estimate the distribution over the same time-period: one day. Thus we use $\Delta t = 1$. From equation 6.3, the distribution should be

$$R_i \sim \mathrm{N}\left(\left(\mu - \frac{\sigma^2}{2}\right)\Delta t, \sigma^2 \Delta t\right). \tag{6.4}$$

By taking our sampled data R_i and computing the standard deviation, we have an estimate for the daily diffusion σ. In finance, this is called the historic volatility (although not all volatility measures are calculated the same way). Next we find the mean of R_i, and estimate the drift term μ as: $\mathrm{mean}(R_i) + \sigma^2/2$. For the MSFT data, we find the daily drift and diffusion estimators to be $\mu = 0.23\%$ and $\sigma = 1.77\%$ per day.

[9] Argument omitted for brevity. See an introductory computational finance textbook, *e.g.,* [Sey09].

A first note on bias. Recall that the Black-Scholes model assumes that dividends are not paid out. This is not true for historic data. For the example, the MSFT data includes four dividend payments of $0.13 each. This causes the closing price to be adjusted downward. Thus equation 6.1 should differentiate between pre- and post-adjusted prices: denominator S_i should be the post-adjusted price for time i, while numerator S_{i+1} should be the pre-adjusted (raw) price for time $i+1$. We found the difference to be insignificant due to dividends being infrequent and small, and thus ignored dividends.

6.4.2 Monte-Carlo Simulation

With estimates for μ and σ, we next consider the distribution of outcomes over the time-period, τ, for which we want to harvest entropy. To generate this distribution, we use a Monte-Carlo simulation of the process in equation 6.3. We discretize τ, which is one day for the MSFT example, into m equal-sized steps of size $\Delta\tau$. Equation 6.3 can then be simulated as Algorithm 2.

Algorithm 2: A Monte-Carlo trial for simulating asset price movements.

for $(0 \leq j < m)$ **do**
- $t \leftarrow T + j \cdot \Delta\tau$
- $Z_t \leftarrow_r \mathrm{N}(0,1)$
- $S_{t+1} \leftarrow S_t \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)\Delta\tau + \sigma Z_t\right)$

We begin at S_T, the last observed price ($29.88 for MSFT), and map a possible trajectory to $S_{T+\tau}$ by stepping through Algorithm 2. Line 2 simply keeps the timestep notation tidy. Line 3 indicates a random variable drawn from a standard normal distribution. This variable is used in the next line, in conjunction with the μ and σ parameters to step the price forward one interval. We then repeat the algorithm N times to generate many independent possible outcomes for $S_{T+\tau}$. Figure 6.1(c) illustrates the first 10 trajectories, while Figure 6.1(d) shows a histogram of outcomes for 100 000 simulations and a bin size of one cent.

A second note on bias. Generally, Monte-Carlo is subject to three types of error. There is the discretization error due to modelling a continuously random process with m intervals. In our case, the exact solution in line 3 of Algorithm 2 is unbiased by discretization error because it is multiplicative. In models other than GBM, the only known solution may be an additive approximation instead (for example, GBM itself could alternatively be approximated by $S_{t+1} = S_t + S_t\mu\Delta\tau + S_t\sigma Z_t$ which is called Euler time-stepping). In this latter case, the error is order $O(m^{-1})$. The second source of bias is using N trials to

estimate some value. In computational finance, the value of interest is the mean of the N outcomes. With both types of error, we reach the often stated total error of Monte-Carlo methods in computational finance: $O(\max(m^{-1}, N^{-0.5}))$ [Sey09]. However in our case, we have an exact solution that eliminates the first term, and we are interested in the entropy of the distribution of N trials, not the mean, resulting in a different bias for the second term. We will discuss how N influences this bias in the next section after we have defined entropy.

The third possible source of bias is the statistical properties of the random number generator used for line 2 of Algorithm 2. We used the default generator in *Mathematica*, which we believe is more than sufficient for Monte Carlo simulations. It creates a seed from sessional information, uses cellular automata to expand the seed into pseudorandom bits, and Box-Muller to transform these into a normally distributed number.[10] We also experimented with a Mersenne twister, sometimes recommended for use in computational finance (*e.g.,* [Sey09]), and it made no significant difference.[11]

Why use Monte Carlo? Given that we have an exact solution for GBM, in equation 6.3, we could eliminate the discretization step and generate $S_{T+\tau}$ values in a single step. Instead, we use time-stepping to create an approach that is easily replicable if one were to swap GBM for a different financial model where an exact solution is not known—*i.e.,* mean reversion or jump-diffusion models. Our aim is to assist the interested reader in studying different models.

6.4.3 Entropy Estimation

We now consider how much entropy is provided in a stock price over the course of a day. We first estimate the Shannon entropy, which provides an average-case measure of unpredictability, as this has the most intuitive appeal and highlights parameter dependence well. We later will consider min-entropy, which provides a worst-case measure. The Shannon entropy of discrete random variable X with probability mass function $p(x)$ is defined as

$$\mathrm{H}(X) \;=\; -\sum_{x} p(x) \log_2 p(x). \tag{6.5}$$

Given the results from the Monte Carlo simulation, we have a list of N possible outcomes for the price of our asset. Denote these outcomes $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$, where P_i is $S_{T+\tau}$ for the i^{th} Monte Carlo trial. We round each outcome to the nearest cent and place it in a

[10] `http://reference.wolfram.com/mathematica/tutorial/RandomNumberGeneration.html`

[11] Linear feedback shift registers, which Mersenne twisters are based on, are a specific type of cellular automata with a "spiral" boundary condition. See [Wol01, pp. 1088].

set of bins for each unique price. Since many trials will yield the same price once rounded, let $\hat{N} \leq N$ be the number of unique outcomes observed (*i.e.,* non-empty bins). Define $\hat{\mathbf{P}} = \langle \hat{p}_j, \hat{P}_j \rangle$, for $1 \leq j \leq \hat{N}$, as the set of pairs where $\hat{P}_j$ is a unique price in $\mathcal{P}$ and $\hat{p}_j$ is the number of times it appears.

To estimate the Shannon entropy in our set $\mathcal{P}$, we compute

$$\mathrm{H}(\mathcal{P}) \quad = \quad -\sum_{j=1}^{\hat{N}} \frac{\hat{p}_j}{N} \log_2 \left(\frac{\hat{p}_j}{N} \right). \tag{6.6}$$

This type of estimator is known as a maximum likelihood, naive, or plug-in estimator [Pan03]. It works by distributing the random variable into bins and estimating $p(x)$ by dividing the number of outcomes in each bin by the total number of outcomes. The goal of this chapter is to estimate the entropy of a closing price, rounded to the nearest cent, which is a discrete random variable. So we use a bin size of one cent.

A third note on bias. Maximum likelihood estimators (MLE) for Shannon entropy are biased. As with any random sampling, some bins may have more values than they theoretically should and others less and this tends to average out as N increases. However for entropy estimation, an empty bin cannot be included in Equation 6.6 because $0 \cdot \log(0)$ is undefined. Instead, empty bins are dropped from the estimate even if theoretically they should be non-empty. This leads to a negative bias for MLE and the entropy is lower than it should be.

In our case, we want a conservative estimate of entropy and so negative biases of this sort are not so troubling. However the bias can be corrected if we can provide an estimate of how many bins have non-zero probability (relative to the number of samples). To estimate this value, we take the full range of $\min(\mathcal{P})$ to $\max(\mathcal{P})$. Let this be $\hat{M}$ bins. The Miller-Madow bias [Mil55] of an MLE is given as

$$B \quad = \quad \frac{\hat{M} - 1}{2N}. \tag{6.7}$$

As stated, it is a negative bias and so the adjusted entropy is: $H_B(\mathcal{P}) = H(\mathcal{P}) + B$. Other adjustments exist in the literature [Pan03]. We selected Miller-Madow because it is computationally easy to compute for large values of N and appropriate when $N > \hat{M}$.[12] For the MSFT data that we have been using to illustrate each step, we used $N = 100\,000$ and found $\hat{N} = 397$ and $\hat{M} = 447$. The MLE Shannon entropy is 7.764 bits and the Miller-Madow bias is 0.002, which is relatively small.

[12]More precisely, if $N/\hat{M}$ diverges to ∞ as N grows, then $H_B(\mathcal{P})$ will converge on the correct result.

Min-Entropy. While Shannon entropy provides an estimate of the average entropy in a stock price, a worst-case estimate is needed if we want to extract the randomness out of the price. Given a distribution $\mathcal{X}$, the min-entropy, $H_\infty(\mathcal{X})$, is defined as

$$\mathrm{H}_\infty(\mathcal{X}) \quad = \quad -\log_2(\max_x(p(x))). \tag{6.8}$$

In other words, for any possible outcome x, $p(x) \leq 2^{-\mathrm{H}_\infty(\mathcal{X})}$. If $\mathcal{X}$ is uniformly random, the Shannon entropy and min-entropy are equivalent. Otherwise, min-entropy is strictly less. Since $H_\infty(\mathcal{X})$ is ultimately computed from one probability $p(x)$ and this value will be non-zero if the entropy is non-zero, the bias from empty bins on Shannon entropy does not apply to the notion of min-entropy.

We use the Shannon entropy estimates to examine the effect of drift, diffusion, initial price, and elapsed-time in the next subsection. We use the min-entropy estimate when we want to configure a random extractor to produce a short, near uniform-random bit-string from the much longer set of prices.

6.4.4 Experimental Results

In the Scantegrity II municipal election, a portfolio of 30 stocks was used.[13] These stocks were the companies that make up the Dow Jones industrial average (DJIA) — an important financial benchmark. Table 6.1 shows the data that we collected and our estimates for each of these stocks following the approach outlined for the MSFT example. The prices were observed from March 23, 2009 to March 23, 2010 (S_0 to S_T), and these were used to estimate the daily drift and diffusion rates: μ and σ. With these parameters, we simulated the path from the closing price on March 23, 2010, S_T, forward one day in time using a Monte Carlo simulation with 100 000 trials. The number of unique prices (to the nearest cent) in our simulation, $\hat{N}$, is used to generate an MLE-estimate for the Shannon entropy: $H(\mathcal{P})$. The estimated number of expected non-empty bins, $\hat{M}$, is used to estimate the Miller-Madow bias: B. These are combined to generate our adjusted estimate of Shannon entropy: $H_B(\mathcal{P})$. Finally, we also provide our estimate of the min-entropy: $H_\infty(\mathcal{P})$. Pfizer (PFE) had the lowest entropy: 6.83 and 6.10 bits for Shannon and min-entropy respectively, while Caterpillar (CAT) had the highest at 9.46 and 8.69 bits respectively.

We were also interested in the individual effect of drift, diffusion, opening price, and time-period on the entropy of a stock. We created a mythical stock with the mean value for each of these parameters, calculated from the DJIA data. The stock had $\mu = 0.195\%$, $\sigma = 1.87\%$, and $S_T = \$48.02$ and was simulated over one day. For each parameter, we

[13] B. Adida. Takoma Park: Meeting 2. `http://benlog.com/articles/2009/11/02/takoma-park-meeting-2/`

Stock	S_T	μ	σ	$\hat{N}$	$\hat{M}$	$H(\mathcal{P})$	B	$H_B(\mathcal{P})$	$H_\infty(\mathcal{P})$
AA	14.50	0.00338956	0.0354406	386	440	7.72544	0.0022	7.73	6.99
AXP	41.24	0.00512444	0.0365912	1071	1305	9.2823	0.006525	9.29	8.50
BA	72.18	0.00313975	0.0219116	1112	1406	9.34279	0.00703	9.35	8.57
BAC	17.13	0.00455058	0.0468486	588	699	8.37453	0.003495	8.38	7.62
CAT	62.41	0.00352696	0.027272	1173	1540	9.4536	0.0077	9.46	8.69
CSCO	26.64	0.00200486	0.0167037	347	396	7.52981	0.00198	7.53	6.79
CVX	74.77	0.000565046	0.0136131	730	844	8.70798	0.00422	8.71	7.95
DD	38.31	0.0025213	0.0219181	603	751	8.43244	0.003755	8.44	7.69
DIS	34.01	0.00271648	0.0199576	506	596	8.13347	0.00298	8.14	7.39
GE	18.33	0.00264998	0.0239698	335	391	7.50284	0.001955	7.50	6.76
HD	32.59	0.00166033	0.0161739	404	475	7.76791	0.002375	7.77	7.03
HPQ	53.15	0.00234904	0.015783	615	758	8.43501	0.00379	8.44	7.69
IBM	129.37	0.00124652	0.0124436	1121	1460	9.36931	0.0073	9.38	8.60
INTC	22.67	0.0019257	0.0176758	302	352	7.37295	0.00176	7.37	6.63
JNJ	65.36	0.00101973	0.00811278	406	472	7.7723	0.00236	7.77	7.03
JPM	44.58	0.00261719	0.0318482	992	1190	9.18918	0.00595	9.20	8.43
KFT	30.49	0.00134673	0.0129888	314	333	7.35464	0.001665	7.36	6.62
KO	55.30	0.0010976	0.0111199	460	570	7.98678	0.00285	7.99	7.21
MCD	67.35	0.00111279	0.0113681	569	732	8.3043	0.00366	8.31	7.55
MMM	82.35	0.00235099	0.0148201	854	1075	8.97369	0.005375	8.98	8.22
MRK	38.50	0.00162879	0.0166847	486	554	8.05935	0.00277	8.06	7.29
MSFT	29.88	0.0022737	0.0176583	394	449	7.76265	0.002245	7.76	7.04
PFE	17.54	0.00120496	0.01571	216	243	6.82701	0.001215	6.83	6.10
PG	64.53	0.00146004	0.0125241	587	703	8.37914	0.003515	8.38	7.64
T	26.55	0.000357228	0.0121909	251	289	7.05851	0.001445	7.06	6.31
TRV	53.90	0.00154645	0.0188065	734	926	8.7059	0.00463	8.71	7.96
UTX	73.09	0.00232501	0.0159515	835	1015	8.9016	0.005075	8.91	8.14
VZ	30.98	0.000367966	0.0117435	279	320	7.22926	0.0016	7.23	6.48
WMT	55.89	0.000497465	0.010295	431	512	7.89168	0.00256	7.89	7.16
XOM	66.95	0.0000317968	0.012391	604	752	8.41962	0.00376	8.42	7.65

Table 6.1: The Dow Jones portfolio of 30 stocks. For the Monte-Carlo parameters, we show: initial price (S_T), historic diffusion parameter (μ), and historic drift parameter (σ). For the Shannon entropy estimate, we show: the number of unique prices in the simulation ($\hat{N}$), the estimated number of non-empty bins ($\hat{M}$), the Shannon entropy estimate based on observed prices ($H(\mathcal{P})$), the Miller-Madow bias (B), and the bias-adjusted estimate ($H_B(\mathcal{P})$). For the min-entropy estimate, we show the estimate based on observed prices ($H_\infty(\mathcal{P})$).

	Drift		Diffusion		Initial Price		Time	
	μ	$H(\mathcal{P})$	σ	$H(\mathcal{P})$	S_T	$H(\mathcal{P})$	τ	$H(\mathcal{P})$
Min	0.003%	6.81	0.81%	8.53	\$14.50	7.33	0.5 day	8.03
Mean	0.195%	8.54	1.87%	8.54	\$ 48.02	8.54	1 day	8.54
Max	0.513%	9.95	4.68%	8.54	\$ 129.37	9.85	2 days	9.03

Table 6.2: This table shows the result on the entropy of the closing price if each parameter is independently changed from its average value across the DJIA to its minimum and maximum.

individually varied it to the minimum and maximum observed value for this parameter in the DJIA data and estimated the resulting Shannon entropy. We also varied the timeframe from half a day to two days. The results are provided in Table 6.2. The entropy was sensitive to the observed spread in both μ and S_T but largely invariant to changes in σ. In all cases, an increase in the parameters resulted in an increase in the entropy.

6.4.5 Portfolio Entropy

We have shown how to estimate the entropy of individual stocks. But how much entropy is in a collection of stock prices? From Figure 6.1, it may be tempting to sum the entropy estimates for all the stocks and use this as an estimate of the total entropy in the portfolio. This approach works only if the stocks are uncorrelated with each other. In reality, stocks typically display varying degrees of correlation with other stocks from, for example, the same business sector, same country, or when traded on the same index. This means that the portfolio entropy is less than the sum of the entropy of the individual stocks due to mutual information between subsets of the stocks.

We selected the two stocks with the highest correlation and estimated the mutual information between them. Pairwise throughout the portfolio, the highest correlated stock pairs are Cheveron and Exxon (0.82 over one year) which are both large oil and gas producers. Next are JP Morgan and Bank of America (0.78) which are both large banks. The mean correlation was 0.42.

Taking Cheveron and Exxon (CVX and XOM), we generated correlated Monte Carlo paths [Sey09]. To do this, recall Algorithm 1. We perform this algorithm for CVX. Denote the value of Z_t in line 3 for CVX as Z_t^C. For XOM, our second stock, we will run almost the same algorithm; we replace line 4. First denote the output at line 3 as Z_t^X and let the correlation between CVX and XOM be ρ. Then the replacement for line 4 is

$$\begin{aligned} S_{t+1}^X \leftarrow S_t^X \exp\Bigg(&\left(\mu - \frac{\sigma^2}{2}\right)\Delta\tau \\ &+\sigma\left(\rho Z_t^C + \sqrt{1-\rho^2}Z_t^X\right)\Bigg). \end{aligned} \tag{6.9}$$

In other words, Z_t^X is used as a joint random variable shared between both stocks.

Using this method, we ran 10 000 000 trials to estimate the joint Shannon entropy between Exxon and Cheveron. This was the largest simulation that was computationally feasible for us. Recall from Table 6.1 that the number of unique prices observed for Cheveron and Exxon were respectively 730 and 604. That means the number of unique price pairs is on the order of 730*604. Thus our simulation of 10M trials was only one order of magnitude greater than the number of observed events and our result is a quite sparse

histogram from which the estimates will be biased. We found that the joint entropy was 15.96 bits compared to the sum of their individual entropies: 16.90 bits. That means the mutual information is at most 0.94 bits. Again, we did not adjust for bias (Miller-Madow is best when the trials are much larger than the outcomes) and so the mutual information is likely less than this. For min-entropy, the result is 1.04 bits.

We leave a rigorous analysis of mutual information in the entire Dow Jones index for future work. As mentioned, computing the joint entropy between two stocks is very difficult as it is: the number of bins squares, as does the number of trials needed to create a suitably dense histogram. Methods exist for estimating bias when the trials are less than the number of bins [Pan03] but it is not obvious how to extend these estimators to more than two random variables. Computing the joint entropy between 30 stocks does not seem computationally feasible, even if the trials could be less than the resulting bins by a polynomial factor.

We can provide a very crude estimate by making a generous assumption. Recall the chain rule for joint entropy is as follows:

$$H(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n) = \sum_{i=1}^{n} H(\mathcal{P}_i | \mathcal{P}_{i-1}, \ldots, \mathcal{P}_1). \tag{6.10}$$

We propose using $H(\mathcal{P}_i) - \max_{(i,j<i)} (I(\mathcal{P}_i, \mathcal{P}_j))$ as an estimate of the $H(\mathcal{P}_i|\mathcal{P}_{i-1}, \ldots)$ term for each stock. This would hold if the worst-case mutual information between any two stocks in the portfolio was less than the mutual information a given stock has with the rest of the portfolio. In other words, any mutual information Cheveron shared with a stock other than Exxon (because of similarities in sectors, country, exchange, *etc.*) would already be accounted for in the mutual information it shares with Exxon. This is crude and the estimate should be treated only as a ballpark figure. For the DJIA under this assumption, the Shannon entropy is 218 bits and min-entropy is 192.

6.5 Beacon Implementation

We have estimated that the closing price for a single stock in the DJIA provides between 6 and 9 bits of entropy, and we have shown that adding additional stocks increases the entropy. In this section, we consider how to convert a list of closing prices into a form that is useful for general cryptographic protocols. Since the field of cryptography has conventions in notation that sometimes conflict with conventions in computational finance or information theory, we will, in a small number of cases, redefine variables in this section.

Let $\mathcal{P}_i$ be a list of closing prices from our portfolio on day i. We encode the prices as integers with a fixed, sufficient number of digits and concatenate them (*i.e.,* $\{14.34, 41.08, \ldots\}$

$\rightarrow 001434\|004108\|\ldots$). Let the bit-length $|\mathcal{P}_i| = n$. If the prices in the list are, for example, the 30 stocks of the DJIA, encoding each price with 6 digits will produce a 180 digit or 598 bit string. However from our simulations in the previous section, we estimate that there would be only 218 bits of entropy in this 598 bit string, and 192 extractable bits.[14] Because the randomness is not concentrated, this semi-random string is not suitable for seeding selection algorithms or pseudorandom generators, or as a cryptographic challenge. In this section, we provide a protocol that, among other things, takes as input a long, semi-random string of stock prices, and output a shorter string of near-uniform random bits.

The stock market prices can be used in at least one of two ways: it can be sampled directly by any party requiring a random challenge, or, alternatively, a neutral third party can regularly sample the source, produce a random bit-string in some useful form, and publish it for quick reference by any party requiring an unpredictable challenge (or seed, common reference string, nonce, *etc.*). We call such a publisher a *beacon service provider* (BSP) and we will refer to the random values it produces as *seeds*. In general, the BSP will be agnostic of what the seeds are being used for.

A BSP can provide a few benefits: if an election (or other protocol) has a low risk of fraud, the fact the entity claims a seed is random may be trustworthy enough. In higher risk protocols, the derivation of the seed can be independently verified and the BSP is not trusted at all. Since the BSP is regularly publishing a new seed, it can update in a recursive fashion, mixing the new closing prices with the previous seed. This makes the output dependent on the prices that preceded it. Finally, as an incentive to provide a beacon service, we can, with care, also allow the BSP to influence the seed with its own randomness. This also provides a marginal hedge for everyone else: even if the closing prices are fully manipulated, the attacker will still have to also collude with the BSP to fix the seed.

6.5.1 Definitions

Randomness Extractor. Briefly, $\texttt{Ext}_k$ is a function: $\{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$. It takes an n-bit input of sufficient entropy and a d-bit key k and returns an m-bit output of high entropy where $m < n$. Here sufficient entropy means the min-entropy of $\{0,1\}^n$ is at least m bits (we will generally require $2m$) and high entropy means that the statistical distance between the distribution on $\{0,1\}^m$ and m uniform random bits is ϵ, where ϵ is a negligible function in m. We only consider the case where $d = m$. For a formal definition, see [DGH$^+$04].

[14] Even if we did not prepend each price with as many leading zeros, the entropy would still be less than the length of the string. This is because the most significant digits in the price is much less likely to change than the least significant digit (*i.e.,* the one cent digit).

CBC-MAC. Briefly, `CBC-MAC` is a mode of operation for a block cipher based on cipher block chaining (CBC) with an initialization vector of zero, but it returns only the final block. It is suitable for creating message authentication codes (MAC). Dodis *et al.* show that `CBC-MAC` is an extractor if the block cipher is an ideal random permutation, the plaintext has $2m$ bits of min-entropy for an m-bit output, and the key is uniformly random. For the full details (including an upper bound on the input size), see [DGH$^+$04].

Pseudorandom Generator. Briefly, `G` is a function: $\{0,1\}^m \rightarrow \{0,1\}^{l(m)}$. It takes an m-bit seed and returns a polynomially-bounded number of bits $l(m)$ typically greater than m. The distribution on $\{0,1\}^{l(m)}$ is ϵ-close to uniform random. For a formal definition, see [BH05].

Robust Pseudorandom Generators. Barak and Halevi provide a construction for a robust pseudorandom generator [BH05]. A robust PRG uses a standard PRG, `G`, as well as maintaining some evolving internal state s_i that can be referenced during operation. `G` is used $\{0,1\}^m \rightarrow \{0,1\}^m$ (*e.g.,* $l(m) = m$) and is assumed to be cryptographically secure: the internal state cannot be inferred from the output by an adversary bound to probabilistic polynomial time (PPT). The construction separates the task of updating the state from the task of generating pseudorandom bits. We only reference the former function: `refresh`.

$$\begin{aligned} \texttt{refresh}: \quad & \{0,1\}^m \times \{0,1\}^n \rightarrow \{0,1\}^m \\ & s_i \leftarrow \texttt{G}(s_{i-1} \oplus \texttt{Ext}_k(\mathcal{P}_i)). \end{aligned}$$

This function is of interest because it could be used for updating the seeds of a beacon service, where the seed is substituted for the state. A robust PRG affords a number of functions to an adversary. Of interest, `badRefresh` lets the adversary refresh with a chosen $\mathcal{P}_i$, and `setState` allows the adversary to learn s_i and replace it with a chosen s_{i+1}. In the face of these attacks, a robust PRG provides three properties: forward secrecy (not needed for a beacon service since past seeds are public), break-in recovery, and resilience. Briefly, break-in recovery means that if an adversary learns s_i for round i, s_{i+1} is still unpredictable if `refresh` is run with a sufficiently random $\mathcal{P}_{i+1}$. Resilience means that if an adversary controls $\mathcal{P}_{i+1}$ but does not know s_i, s_{i+1} is unpredictable.

Bilinear Groups. Let $\mathbb{G}$ and $\mathbb{G}_T$ be cyclic groups of order q, where q is a large prime. Let $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ be a bilinear, non-degenerate, and efficient mapping. That is, $\forall g_1, g_2 \in \mathbb{G}$ and $\forall a_1, a_2 \in \mathbb{Z}_q$: $e(g_1^{a_1}, g_2^{a_2}) = e(g_1, g_2)^{a_1 \cdot a_2}$ where e is computable in polynomial time and $e(g, g)$ generates $\mathbb{G}_T$ for some $g \in \mathbb{G}$. Finally, assume the q-decisional bilinear Diffie-Hellman inversion assumption holds for the group (definition omitted for brevity – see [BF05]).

Verifiable Unpredictable Function. Briefly, a VUF is a tuple of four functions. `GenK` returns secret key sk and public key pk. $\texttt{Eval}_{sk}(x)$ takes input x and returns a unique and unpredictable y (we omit defining the domain and range, as they vary by construction). $\texttt{Proof}_{sk}(x, y)$ returns proof $\pi_{x,y}$ that y is the output on x. $\texttt{Verify}_{pk}(x, y, \pi_{x,y})$ returns 1 iff the proof is correct. A VUF is a weaker primitive than a verifiable random function (VRF). Given x, a VUF guarantees y cannot be guessed with non-negligible advantage, while a VRF offers the stronger guarantee that y cannot be distinguished from a random element with non-negligible probability. A VUF is sufficient for us because we know the entropy in y and will use an extractor to convert it into the required form. For a formal definition of both, see [DY05].

Dodis-Yampolskiy VUF. Dodis and Yampolskiy propose the following VUF, defined over a bilinear group: $\{\mathbb{G}, \mathbb{G}_T, g \in \mathbb{G}, q\}$ [DY05]. `GenK` publishes the group, randomly generates secret key $sk \leftarrow_r Z_q^*$ and publishes public key $pk = g^{sk}$. $\texttt{Eval}_{sk}(x)$ takes $x \in Z_q^*$ and returns y, which in practice is a point on an elliptic curve. $\texttt{Eval}_{sk}(x)$ is defined as $y \leftarrow g^{\frac{1}{x+sk}}$. The proof is embedded in y; $\texttt{Proof}_{sk}(x, y)$ simply returns $\{x, y\}$. $\texttt{Verify}_{pk}(x, y, \pi_{x,y})$ is defined as $e(pk \cdot g^x, y) \stackrel{?}{=} e(g, g)$.

6.5.2 Security Properties

The central function of our BSP protocol updates the seed at interval i with new randomness $\mathcal{P}_i$. We define it with the recurrence: $s_i \leftarrow \texttt{Update}(s_{i-1}, s_{i-2}, \mathcal{P}_i)$, where s_i is the seed at interval i and s_{i-1} and s_{i-2} are the two preceding seeds. An update function does not need to use *two* previous seeds—it could be more or less. The seeds are akin to the internal state of a robust PRG, except that they are made public at each interval. We assume the BSP has a secret key. We have several properties we want from our protocol.

- **Verifiable.** A polynomial time verifier should be convinced that `Update` was computed from $\mathcal{P}_i$, s_{i-1}, and s_{i-2} as specified in the protocol.
- **Statistically Random.** The statistical difference between s_i and a uniform distribution of the same length should be ϵ-close.
- **Unpredictable.** The advantage of a PPT-bound adversary in predicting s_i without $\mathcal{P}_i$ is negligible, even if the adversary knows s_{i-1}, s_{i-2} and the BSP's secret key.
- **Parisian.** The advantage of a PPT-bound adversary in computing the exact value of $\mathcal{P}_i$ needed to output a chosen s_i is negligible if the target s_i must be specified before the adversary knows $\mathcal{P}_{i-1}$, s_{i-1} and s_{i-2}, even if she knows the BSP's secret key.
- **Partially Distributed.** The advantage of a PPT-bound adversary in predicting s_i is negligible if she knows s_{i-1} and s_{i-2}, chooses $\mathcal{P}_i$, but does not know the BSP's secret key.

	Variant	Verifiable	Unpredictable	Statistically Random	Parisian	Partially Distributed	Output Length ($\lvert s_i \rvert$)
1.	$s_i = \mathcal{H}(\mathcal{P}_i)$	x	x				$\{0,1\}^h$
2.	$s_i = \texttt{Ext}_k(\mathcal{P}_i)$	x	x	x			$\{0,1\}^m$
3.	$s_i = \mathcal{F}_{\text{owf}}(s_{i-1} \oplus \texttt{Ext}_k(\mathcal{P}_i))$	x	x	x	x		$\{0,1\}^m$
4.	$s_i = \texttt{Ext}'_{k'}(\texttt{Eval}_{sk}((s_{i-1} \Vert s_{i-2}) \oplus \texttt{Ext}_k(\mathcal{P}_i)))$	x	x	x	x	x	$\{0,1\}^l$

Table 6.3: A sequence of intermediate protocols, and which properties they achieve, leading to our suggested protocol. Assuming $\mathcal{P}_i$ has $2m$ bits of min-entropy, the output length is provided. Length h depends on the hash function used, while $m = 2l$ (*e.g.,* h=160, m=256 and l=128).

With the exception of the Parisian property, the motivation for these is hopefully intuitive and not in need of further explanation. We will illustrate the last property with a concrete example. Suppose on the Tuesday night of a given week, election officials post the results of an election and the post-election challenges are to be generated on the Friday of that week after the markets close. On Thursday night, the adversary knows the seed from Thursday and let us assume she can fully manipulate Friday's closing prices. Further, assume that she has a particular seed that she wants produced to ensure some fraud committed on Tuesday escapes detection. It should be computationally infeasible for the adversary to determine the closing prices she needs to set in order to generate, from the existing seeds, this target seed she wants. The only way the adversary can produce the desired seed on Friday is to manipulate the prices every day: Wednesday, Thursday, and Friday. We name this "parisian" after the Parisian options we discussed earlier, which required a barrier event to occur on multiple days.

6.5.3 Protocol

Variants. In this section, we define a protocol that can be used by a BSP to implement the `Update` function described in the previous section. The stock market randomness, $\mathcal{P}_i$, is n bits and contains $2m$ bits of min-entropy. We reference two extractors with different output sizes: $\texttt{Ext}_k$ has an m-bit key and produces an m-bit output, while $\texttt{Ext}'_{k'}$ has an l-bit key and l-bit output. We consider $m = 2l$.

Table 6.3 shows some related constructions. Perhaps the simplest approach is to publish an h-bit hash of the list of closing prices each day: Variant 1. Before $\mathcal{P}_i$ is known, the hash of $\mathcal{P}_i$ will trivially be unknown as well, and once $\mathcal{P}_i$ is known, anyone can verify the output is correct by recomputing the hash from their source for the closing prices.

The problem with using a fixed hash function is that it does not guarantee a statistically random output. Dodis *et al.* show that even if a hash function is modelled with an ideal compression function (and Merkle-Damgaard chaining), it does not have good extraction properties. Instead, the hash needs to be taken from a family of hash functions (*i.e.,* a keyed hash) and even then, one must pay attention to the padding scheme used to ensure the final block has sufficient entropy [DGH$^+$04]. With the use of a proper extractor, Variant 2 of the protocol produces a statistically random output. While extractors are keyed, the key, k, only needs to be uniformly random. Its value is not kept secret and the key can be reused.

Variant 2 is not Parisian because it is only functionally dependent on the most recent closing prices of the stocks. Let $\mathcal{F}_{\text{owf}}$ denote a one-way function. Variant 3 adds the Parisian property and is essentially the `refresh` function of a Barak-Halevi robust PRG. Recall `refresh` is defined as: $s_i \leftarrow \mathtt{G}(s_{i-1} \oplus \mathtt{Ext}_k(\mathcal{P}_i))$. `G` is used as: $\{0,1\}^m \rightarrow \{0,1\}^m$ (*i.e.,* there is no expansion of m) and PRGs are one-way. [15]

Variant 4 is our protocol. The main modification is to replace $\mathtt{G}(\cdot)$ with the $\mathtt{Eval}_{sk}(\cdot)$ function of a VUF, which allows the BSP to influence the seed with sk, while maintaining verifiability. This adds the partially distributed property. The other variants were verified by recomputing the value. In this case, the BSP produces a correctness proof for $\mathtt{Eval}_{sk}(\cdot)$ that is checked. The properties of the VUF also imply that without knowledge of sk, the output of $\mathtt{Eval}_{sk}(\cdot)$ is unpredictable if the input is unknown.

VUF Details. A variety of VUFs and VRFs exist. We use the Dodis-Yampolskiy VUF [DY05] for its simplicity, efficiency, and the fact that its proof is non-interactive.[16] However since it is based on bilinear pairings, we require some encoding. Recall the domain of `Eval` is an exponent and the range is a point on an elliptic curve. Let Φ_1: $\{0,1\}^m \rightarrow \mathbb{Z}_q^*$ be an encoding function that is entropy preserving (*i.e.,* injective). When $q > 2^m$, this encoding is the trivial one. Let Φ_2 be a mapping from an element in $\mathbb{G}_T \rightarrow \{0,1\}^m$. This encoding (or extraction) is more difficult, and it is specific to the type of elliptic curve used. For a random point on an ordinary curve defined over $\mathbb{F}_{2^{2l}}$, an l-bit string can be extracted [FPS08]. For a supersingular curve (which offer fast pairing operations), we are not aware of any specific extractor. Instead, we simply concatenate the coordinates together and use a randomness extractor, $\mathtt{Ext}'_{k'}$, which produces an l-bit output.

Extractor Details. In deciding on how to implement $\mathtt{Ext}_k$, we consider two options.

[15] So why is `G` not a one-way function in the Barak-Halevi model? The answer is because we are only using one of two functions offered by the robust PRG model—the other function uses the same G as: $\{0,1\}^m \rightarrow \{0,1\}^{2m}$.

[16] Note that while Dodis-Yampolsky restrict the domain of their VUF, Camenisch *et al.* find it can admit the full range under a stronger security assumption (in the generic group model) than q-BDHI [CHK$^+$06].

- We could use an approach that is specific to the distribution we are drawing from: closing prices (for the first extraction). For example, we could consider a daily relative increase in price as heads and a decrease as tails. Due to the drift term, there is a slightly biased toward heads which can be corrected for with a Von Neumann extractor—see Footnote 1. However this approach produces less than a bit of entropy per closing price (and therefore does not guarantee even one bit of entropy each day the market closes) and since stocks are correlated, it is not clear how to use more than one stock.
- Dodis *et al.* investigate using a block cipher in CBC-MAC mode—the result quoted above. This construction has less "fine-print" than some of the alternatives they examine. If the plaintext has $2m$ bits of min-entropy and is L blocks long, it produces an m-bit output with a statistical distance, from a uniformly random m-bits, of $\mathcal{O}(L\ 2^{-m/2})$. A side-benefit is that it also has an "avalanche-effect" where a difference in a single bit of the input causes a random-looking difference in the output. We use this approach.

Key Derivation. The CBC-MAC extractor, or any other non-deterministic extractor, does have one issue however: how it is keyed. Extractor keys do not need to be secret (in fact, for verifiability, they cannot be), however they must be uniformly random. Again, we considered a few options.

- The BSP could choose a uniformly random key at the beginning of the protocol and use it throughout. However this is providing adversarial control over the key, and we cannot cite any strong results on what an adversary could do with this control.
- Assuming we can bootstrap the process, we will be generating good quality randomness at every iteration and so we could use the values of the previous seeds to refresh the key to the extractor. The results from the extraction literature assume a uniform random key and do not indicate if an ϵ-close random key is sufficient (or could be compensated for by increasing the entropy of the input).
- We could use the historic prices of a single stock and a Von Neumann extractor as described in the first bullet point of the preceding list. While we rejected this approach for the extractor itself, it works well for generating a key. Since there are no unpredictable or secrecy requirements on the key, we can use past prices instead of future prices. This gives us immediate access to enough bits to generate the two required extraction keys (k and k'). In addition, the key can be continually updated by shifting in new bits as time goes by. We give full details of this key derivation procedure in Appendix B.

Algorithm 3: Update Beacon

begin

$x_i = (s_{i-1} \| s_{i-2}) \oplus \texttt{Rijn256-CBC-MAC}_{k_i}(\mathcal{P}_i)$

$y_i = g^{\frac{1}{x_i + sk}}$

$s_i = \texttt{AES128-CBC-MAC}_{k'_i}(y_i)$

Publish: $\{y_i, s_i\}$

Algorithm 4: Verify Beacon

begin

$x_i = (s_{i-1} \| s_{i-2}) \oplus \texttt{Rijn256-CBC-MAC}_{k_i}(\mathcal{P}_i)$

$e(pk \cdot g^{x_i}, y_i) \stackrel{?}{=} e(g, g)$

$s_i \stackrel{?}{=} \texttt{AES128-CBC-MAC}_{k'_i}(y_i)$

Protocol. The BSP publishes pairing-friendly $\{\mathbb{G}, \mathbb{G}_T, g \in \mathbb{G}, q\}$. The BSP chooses sk randomly from Z_q^* and publishes $pk = g^{sk}$. The BSP generates the current extraction keys k_i and k'_i. The initial states s_{-1} and s_0 are set to zero. On day $i \geq 1$, BSP takes the closing stock prices, P_i, and executes Algorithm 3. To verify that s_i is a proper update to s_{i-1}, a verifier with $\{pk, k_i, k'_i, \mathcal{P}_i, y_i, s_i, s_{i-1}, s_{i-2}\}$ executes Algorithm 4. Note that the verifier does not need to verify that the seeds from s_{i-1} back in time to s_1 were themselves correctly formed. To be assured that s_i is a random beacon, it is sufficient to check it is correctly formed from the arbitrary values claimed to be s_{i-1} and s_{i-2}. This is because $\mathcal{P}_i$ fully refreshes the randomness of s_i.

Parameter Sizes. For the extractors, we need a block cipher and AES is the default candidate. However we also require the block size and key length to be the same; thus, only AES-128 is suitable. If we use Rijndael, we can expand the choice of message block length to 128, 192, or 256 bits. Our protocol requires two extractors and with each extraction, we lose half of the bits of entropy in the input. Therefore we start with 512 bits of min-entropy in $\mathcal{P}_i$ and use Rijndael-256[17] to generate a value x_i with (ϵ-close to) 256 bits of min-entropy. These 256 bits are preserved in y_i and then a second extraction with AES-128 produces a value s_i with (ϵ-close to) 128 bits of min-entropy.

This approach depends on $\mathcal{P}_i$ having at least 512 bits of min-entropy. We estimated the DJIA only has 192 bits. However we have also shown that adding additional stocks increases the total min-entropy. We would like the portfolio to be easy to construct, and ideally iconic—for this reason, we recommend using the S&P 500. By extrapolating our results, the daily closing prices of these 500 stocks provides 512 bits of min-entropy with a

[17]Rijndael with a 256 bit block size and a 256 bit key.

large safety margin.[18] This will allow us to produce a fresh 128-bit random number every market day.

This approach also assumes we can input the 256-bit number x_i into our VUF, implemented in the subgroup of a pairing-friendly curve, without losing entropy. The encoding itself is direct for our VUF: x_i is input into an exponent. The issue is the size of the subgroup as common curve sizes work in subgroups of 160-bits, so our approach would require a custom implementation. If alternative sizes for the extractors are preferable, one could use a variable-length block cipher [AB96] instead of Rijndael/AES, which allows extractors of any size.[19]

6.5.4 Security Analysis (Abstract)

We omit the analysis of security for our protocol; however, it is included in Appendix B. In the analysis, we prove that our protocol has the five properties we have specified. We note that some of the properties we want to prove are subsumed by the properties of a Barak-Halevi robust PRG. This includes directly: *statistically random* and *unpredictable* (unpredictable is equivalent to their notion of break-in recovery). *Parisian* is also implied but not explicit. However, we prove each of these independently for our variant, plus demonstrating *verifiability* and the *partially distributed* property.

6.6 Concluding Remarks

In this chapter, we have shown that the closing prices for common stocks contain sufficient min-entropy for generating random challenges for use in elections or other cryptographic applications. This result, in combination with the ease with which stock prices can be verified, makes financial data an attractive source of randomness for cryptographic election systems unable to use the Fiat-Shamir heuristic, and possibly for precinct selection in standard electronic voting. In addition to the entropy estimates, we have provided a provably secure protocol for implementing a beacon service. It is our hope that such a beacon service, whether using our protocol or a variant, becomes a reality.

Future Work. We list a few items for future work. First, estimating joint entropy between correlated stocks becomes infeasible as the number of stocks grows. Progress on

[18] This is not meant to suggest that the 30 stocks used by Scantegrity are insufficient. For the number of units to be audited in this election, a seed of 16 bits expanded with a PRNG provides good statistical properties (*cf.* [CEA07, Res09]). As shown, this is nearly achieved by CVX and XOM alone.

[19] Although the *security* of these ciphers has not been examined as closely as AES, neither has been examined in any detail for good *extraction* properties.

making this estimation more efficient would be welcome. Second, some results concerning extractors built with standard cryptographic primitives would be useful: specific bounds (instead of asymptotic bounds), analysis of using inputs where the min-entropy is less than twice the min-entropy of the output, and analysis of extraction with a key that is only statistically-close to uniform random, instead of being exactly uniform random. A third item for future work would be alternative VUFs, in particular a scheme that works over the integers, to reduce the complexities of mapping from a finite field over elliptic curves back to the integers. Ideally, if a VUF mapped m-bit integers to m-bit integers, then the second extractor would not be needed at all.

Chapter 7

Eperio: Lightweight Election Verification in a Spreadsheet

This chapter references and excerpts from portions of published work co-authored with Aleks Essex, and supervised by Urs Hengartner and Carlisle Adams [ECHA10]. The current presentation places more emphasis on personal contributions to the project, although each aspect of the project is to some extent joint work.

7.1 Introductory Remarks

In this chapter we present Eperio, a protocol for cryptographic election verification. We approach the design of Eperio with two primary goals. First, the system should be secure under reasonable assumptions, while using cryptographic primitives that are simple and efficient. Second, the verification software of the desired system should be implementable, wherever possible, using either familiar software tools, or custom open-source software with an emphasis on compact code size.

In its basic form, verifying an election with Eperio involves an auditor downloading a set of encrypted audit files from an election website, opening a subset of them by entering a password and comparing the files to one another for consistency using a spreadsheet application or small software script. We believe this can greatly simplify the typically in-depth technical component of cryptographic election verification, especially relative to related systems. Eperio is design as a drop-in replacement for existing verification mechanisms in paper-based E2E systems (see Section 3.4.3).

Motivation. Of all the requirements we presented in Section 2.2, one of the least studied is understandability. While end-to-end verifiable election systems offer full transparency

by producing a verifiable transcript of the election, for most systems, the argument for why the verification steps actually prove the properties of the system can hardly be called transparent. This touches on a broader debate within usability and human-computer interaction: to what extent do users need a proper mental model of the system to use it correctly?

In our experience, cryptographic election verification faces criticism and resistance engendered by an ostensible lack of understandability as a result of the use of cryptography. In response, E2E systems like Scantegrity (see Section 4.3) have aimed at making ballot casting as close to traditional optical scan as possible. Ballot verification is completely optional, and if voters do opt-in, verifying confirmation codes through a web interface is akin to tracking a parcel. However to fully verify the election, one must also check the consistency of the election data.

During the Scantegrity municipal election in the United States (see Section 4.4), the two independent auditors of the election, both with PhDs in computer science, each wrote several hundred lines of software code to pore over the two and a half gigabytes of cryptographic audit data.[1] Many of the other systems in the literature (see Chapter 3) employ advanced cryptographic primitives and techniques not typically found in standard software libraries. This often results in the cryptographic software components of such protocol needing to be custom coded.

Contributions. This raises two questions: who can *do* the verification and who can *understand* the verification of a cryptographic election? In this chapter, our focus is on expanding the number of people in both of these groups. It is difficult to directly measure our success, especially in the second case. However what we have succeeded in producing is a provably secure verification protocol that:

- can be performed with a very small verification script,
- requires no external libraries or programs that are not in a vanilla distribution of Linux (in fact, it can be run from a Live CD),
- can alternatively be performed manually in a spreadsheet with no code,
- relies on a single cryptographic primitive (a commitment function),
- produces much smaller audit datasets than Scantegrity or other deployed systems, and
- is an order of magnitude faster to verify than Scantegrity and two orders of magnitude faster than deployed public-key based systems.

[1] Software and audit results available online at `http://github.com/benadida/scantegrity-audit/` and `http://zagorski.im.pwr.wroc.pl/scantegrity/`

We contend that universal verification—the ability of anyone to participate in the election audit—is fundamental to the spirit of cryptographic election audits, and that lowering the technical complexity of the audits follows in that spirit. While we are under no delusion that our approach will open cryptographic election verification to everyone (it is also the case that some citizens do not fully appreciate procedures, *e.g.,* statistical recounts or ballot counterfoils, used to add some level of integrity to current elections), we argue that Eperio is a significant step in this direction.

What about hand counts? One may counter that hand-counted paper ballots are ideal: it is an understandable process with some level of integrity. However tallying by hand can be time-consuming for the long, multi-contest ballots common in many US precincts. Furthermore, while offering greater transparency than electronic voting, full confidence in an election result would require observing the process end-to-end, which is a large time commitment, extends to only a single polling place, and is nullified if the ballots are recounted at a later date. We seek a solution that offers full confidence for all precincts and whose audit can be conducted at any convenient time after the election using commonplace computer software.

7.2 Security Model

In general we will require an end-to-end election verification protocol to be secure against two principal threats:

- **Manipulation**. An active adversary may attack the administrative interface, attempting to present a false election outcome by corrupting software components, data, or a tolerable threshold of election officials.
- **Identification**. A passive adversary may attack the verification interface, attempting to use data on the public bulletin board (which persists indefinitely) to recover the unique associations between ballot receipts and individual votes.

If it is successful at preventing these threats, we say it has integrity and ballot secrecy. Eperio is a verification mechanism that can interface with a number of common ballot cast procedures (Punchscan, Scantegrity, Prêt à Voter, *etc.*). Since we are not innovating on the voting interface level, we assume the existence of a voting interface that can interface with Eperio and is secure against the following two threats:

- **Coercion**. An active adversary may attack the voting interface, attempting to coerce a voter into voting in a particular way and producing evidence of compliance.

- **Spurious Disputes**. An active adversary may attack the voting interface, attempting to discredit the validity of the election proof by submitting a false (i.e., spurious) dispute of the results of any of the proofs.

Computational Abilities. Either integrity or ballot secrecy can be achieved unconditionally (but not both in elections with a tallying authority). Everlasting privacy implies that the cryptographic transcript created during the course of an E2E election does not uniquely encode which candidate was voted for on a given receipt and in theory could be any of the candidates. In this case, even a computationally unbounded citizen cannot discover how anyone voted, however a computationally unbounded election authority could change votes without detection. Conversely, unconditional integrity guarantees that the transcript uniquely encodes each vote. This means that probability of detecting a cheating election authority is invariant to its computation power, however ballot secrecy is lost if an observer is computationally unbounded. While we expect that both parties are computationally bounded, offering unconditional security in one regard can eliminate assumptions.

Arguments in favour of everlasting privacy note that robustness only needs to be maintained for the period of the election, while ballot secrecy needs to remain secure against the computational powers of future generations. In the least, secrecy should be post-quantum.[2] On the other hand, the amount of true randomness needed to truly achieve everlasting privacy is inhibiting, especially if secrets are threshold-shared among a set of election trustees. Furthermore, everlasting privacy offers no protection against election trustees colluding or revealing their shares. Eperio can achieve either but we devote most of our explanation to the variant with unconditional integrity.

Integrity Chain. Recall the framework for E2E systems from Section 3.4.1. Slightly restated, it models integrity as the assurance of three sequential properties:

(a) Marked-as-Intended: an auditor can verify that if a voter were to attempt to obfuscate their vote, it would be obfuscated correctly,
(b) Collected-as-Marked: a voter can verify that the obfuscation of their vote is included in a collection of obfuscated votes, and,
(c) Counted-as-Collected: an auditor can verify that the collection of obfuscated votes is properly deobfuscated, producing a self-consistent and correct tally.

[2]Some problems that are assumed to be intractable on classical computers, such as integer factorization and solving discrete logarithms, have known efficient quantum algorithms. Post-quantum cryptography relies on intractability assumptions thought to be preserved with quantum computing.

The composition of these properties results in the voter's preference being *counted-as-intended* (*i.e.,* verifiable from end to end). These properties are realized by three interfaces. A *voting interface* is used by voters on election day to create obfuscations of their votes. A *trustee interface* is used by a set of election trustees to generate the ballots prior to the election, collect the obfuscated votes after the election, and generate a provably correct tally after the election. Finally, a *verification interface* is used by voters and verifiers to check that the proofs are correct. The focus of this paper is on the protocol for verifying this proof (the verification interface), which is somewhat orthogonal to the issue of how the data is generated (the trustee interface) or how ballots are represented (the voting interface).

Trustee Interface. Election trustees construct the cryptographic proofs of integrity using private function evaluation, where no trustee is permitted to learn more from the public outputs than what they can learn from their own private input. We do not assume that the function is evaluated correctly (this will be established with a proof), only that intermediate values are unobservable. We refer to this as a *blackbox* computation. It can be accomplished with a distributed multi-party computation or special hardware. In Section 7.5, we suggest a deployment option that has been used in actual cryptographic elections based on trustworthy computing, but for now leave it as an assumption. We also note that assuming integrity but not privacy, the counterfactual, is also common in the broader cryptographic literature; *e.g., honest-but-curious* adversaries and *tamper-proof* hardware [Kat07, MS08].

The Voting Interface. Although the voting interface is not part of Eperio, it will facilitate discussion of Eperio to discuss various means by which a voter can mark a ballot and construct a receipt. The Eperio protocol works directly with most of the permutation and code-based obfuscation techniques in the literature. For illustrative purposes, we will present how Eperio would work with a paper ballot system similar to the Prêt à Voter system [CRS05]. The list of candidates is independently shuffled on each ballot. The voter finds their preferred candidate and marks the associated position. The voter then tears off just the marks (a figure in the next section, Figure 7.1, illustrates this), which is scanned and then retained as a receipt. The candidate list is shredded by the voter before leaving the polling place. We refer to each position (bubble) on the ballot that can be marked as a unique markable region (UMR).

The Verification Interface. A transcript of certain election events and the scanned receipts are published on an append-only broadcast channel called the *bulletin board*. A robustness proof is generated from the transcript and receipts by a set of election trustees.

The standard approach to verifying the election is through three distinct audits, one for each of the previously mentioned properties (a,b, and c) in the integrity chain:

- **Receipt Check**: Property (b) is verified by checking that a privacy-preserving encryption (or obfuscation) of a vote is present on the bulletin board. A standard method involves granting the voter with a paper (or electronic) receipt of this encryption at the time the ballot is cast. The voter may later use this receipt to consult the bulletin board to ensure it is present and unmodified,
- **Print Audit**: Property (a) is verified by checking that ballots are being presented honestly. That is, a proof that a mark made for a particular candidate will in fact translate into an obfuscated vote for that candidate. A standard method involves granting the voter the option to audit a ballot instead of voting on it. The digital representation of an audited ballot will be de-anonymized so that it can be compared to what was printed (or presented electronically to the voter),
- **Tally Audit**: Property (c) is verified by checking a cryptographic proof of self-consistency of the bulletin board. A standard method involves proving a privacy-preserving mapping from the list of receipts to the election outcome.

Types of Verifier. At first glance it may seem necessary for average voters to understand the underlying cryptographic protocol in order to meaningfully participate. Although ideally every participant would possess such a knowledge base, realistically we see it as a *collaboration* between three user types: voter, election auditor and protocol analyst, each of which is associated with different knowledge and technical requirements.

- **Voter.** A voter is someone who casts a ballot (and receives a receipt) in an election. The primary responsibility of a voter with regard to E2E verification is to substantiate their contribution to the election using their ballot receipt. At a technical level this requires a voter to confirm the election's audit dataset is consistent with their ballot receipt. Conceptually they should be aware of the E2E properties and what role checking their receipt plays in the broader verification process. A voter may also participate as an election data auditor and/or protocol analyst, or delegate it to someone they trust. Alternatively they may choose to ignore the verification process altogether.
- **Election Data Auditor.** An election data auditor undertakes to substantiate the outcome of a *particular* election using the E2E protocol and the public audit dataset for that election. At a technical level they must be able to perform each of the audits either by using existing verification software, or by creating it themselves based on a protocol specification and be able to interpret the results. If using existing verification

software, they must trust its correctness (or review the source code). They must understand the technical details of each audit and be convinced, at a high level, that the protocol is sound (i.e., convincing).

- **Protocol Analyst.** The role of the protocol analyst is to substantiate the security properties of the E2E protocol to the voters and elections auditors. They must possess sufficient expertise to decide at a formal level whether the protocol is complete, sound and secret.

In this chapter, we target the second class: Election Data Auditors. This may encompass software/security enthusiasts, democracy advocacy groups, third party consultants, and political campaigns. For the purposes of this chapter, we present Eperio as an attempt to simplify the technical and knowledge requirements for Election Data Auditors, and make this class of verification more accessible to individuals who would otherwise be confined to the Voter level. Eperio simplifies the technical requirements of the election data auditor through smaller data and code sizes, execution times, and wider implementation options.

7.3 The Eperio Protocol

Eperio is closely related to the Punchscan and Scantegrity cryptographic election verification protocols. Unlike Punchscan and Scantegrity which use a mixnet-like structure to achieve the E2E integrity and privacy properties, Eperio combines its audit data into a single cryptographic table structure. This in turn permits a coarser, more efficient, cryptographic commitment scheme. It also facilitates interesting implementation options such as cryptographic commitments based on file-encryption and E2E verification in a spreadsheet.

Eperio is based on our earlier protocol Aperio. Aperio is a non-electronic, non-cryptographic E2E protocol designed for developing nation-states without a technical infrastructure [ECA08, ECA10]. Aperio achieves E2E verifiability with physical security assumptions. In the following sections, we will translate these assumptions into a cryptographic setting.

7.3.1 Protocol Sketch

As a brief overview of the protocol, a set of trustees will jointly generate a table with three columns. The first column will contain a unique reference for each markable region (*e.g.*, optical scan bubble) on each ballot in the election. The second column will indicate whether a given region was marked or not marked (or alternatively if the ballot was selected for an audit). Finally the third column will contain the candidate/choice associated with each markable region.

The rows of this table are randomly shuffled (analogous to shaking a ballot box). It is easy to see that if the first two columns are revealed, the information should correspond to the set of all of the receipts in the election. If the last two columns are revealed, the information should correspond to the final tally. If all three columns are revealed (or the contents of an unrevealed column are implied through some functional dependency), then ballot secrecy is compromised. The Eperio protocol proves the correct formation of all three columns by only revealing the information implied in two of the three columns. It uses a composition of cut-and-choose and random audit techniques inspired by randomized partial checking [JJR02].

7.3.2 Entities

The Eperio protocol relies on the following entities, which are standard in most E2E voting systems:

- A set of n **election trustees** (or the prover), $\mathcal{P}$, tasked with generating a verifiable tally. $\mathcal{P}$ is assumed to be a set of mutually distrustful and non-collusive trustees. However the protocol can tolerate $t \leq \frac{n-1}{2}$ trustees who collude or refuse to participate.
- The set of authenticated **voters** who cast ballots in the election.
- The first set of **verifiers**, $\mathcal{V}_1$, who verify that their receipts were collected correctly (collected-as-marked). $\mathcal{V}_1$ are either voters, or auditors that were given access to a copy of a voter's receipt.
- The second set of **verifiers**, $\mathcal{V}_2$, who verify that the ballots are printed correctly (marked-as-intended). $\mathcal{V}_2$ are either voters or auditors who went in person to obtain a ballot to audit.
- The third set of **verifiers**, $\mathcal{V}_3$, who verify that the tally is computed correctly from the collected receipts (counted-as-collected). $\mathcal{V}_3$ can include anyone in any location with access to the election data.
- A malicious **adversarial prover**, $\mathcal{P}'$, who will attempt to convince the verifiers that an incorrect tally is correct.
- A malicious **adversarial verifier**, $\mathcal{V}'$, who will attempt to break voter privacy and determine which candidate was selected on a given receipt (or any non-negligible information about this selection).

7.3.3 Functions

The Eperio protocol requires a set of standard functions from the cryptographic literature: a threshold key agreement modeled after the one due to Pedersen [Ped91], which has

been suggested for use in voting by Benaloh [Ben06]; a cryptographically secure PRNG with properties such as the one due to Blum, Blum, and Shub [BBS86]; a perfect shuffle algorithm with properties such as the one due to Fisher and Yates [Dur64]; a commitment scheme with properties such as the one due to Naor (perfectly binding) [Nao89] or Pedersen (perfectly hiding) [Ped92]; and a public coin to generate challenges such as the one due to Clark and Hengartner (Chapter 6). These are merely examples and generally stronger primitives than we envision being used (but they facilitate the security proof in the next section). Section 7.5 suggests alternatives aimed at optimizing efficiency and conceptual simplicity. Note that the PRNG and commitment functions may require additional public and private parameters not specified, depending on the exact implementation used.

Distributed Key Generation/Reconstruction

- $(y_1, \ldots, y_n, \kappa) \leftarrow \mathsf{DKG}(n, t, l, f_1, \ldots, f_n)$
- $\kappa \leftarrow \mathsf{KeyRec}(y_1, \ldots, y_{t+1})$

DKG accepts from each of the n trustees a polynomial of degree t, f_i, with coefficients of bit-length l such that they are elements of $\mathbb{Z}_q$ for a suitably-sized prime q. The coefficients are summed together, mod q, to produce a new polynomial $\hat{f}$. Each trustee i receives $y_i = \hat{f}(i)$ as their share and the value $\kappa = \hat{f}(0)$ forms a shared secret of bit-length l. t should be set such that $n \geq 2t+1$. KeyRec accepts at least $t+1$ shares, $y_1, \ldots, y_{t+1}$, from a subset of the trustees and outputs the shared secret κ.

Pseudorandom Number Generation

- $\{0,1\}^l \leftarrow \mathsf{PRNG}(\kappa, l)$

PRNG is a stateful function which takes as input the shared secret, κ, as a seed and returns l new pseudo-random bits each time it is invoked. For simplicity, we omit l if the size of the output is clear from the context.

Permutation

- $\pi(\mathbf{W}_i) \leftarrow \mathsf{Permute}(w_1, \ldots, w_u, \Pi)$
- $\pi(\mathbf{W}_i) \leftarrow \mathsf{PermuteBlock}(w_1, \ldots, w_u, s, \Pi)$

Permute accepts as input a list $\mathbf{W}_i$ of u elements and a number, $\Pi = \Omega(u \log(u))$, of random bits sufficient to perfectly shuffle the list[3] and returns a list containing

[3]Generating a random integer from random bits is non-deterministic when the integer is not a perfect power of 2. Perfect shuffling algorithms, like Fisher-Yates, require random integers. An upper bound on the *expected* number of bits is $2(\log_2 u!)$.

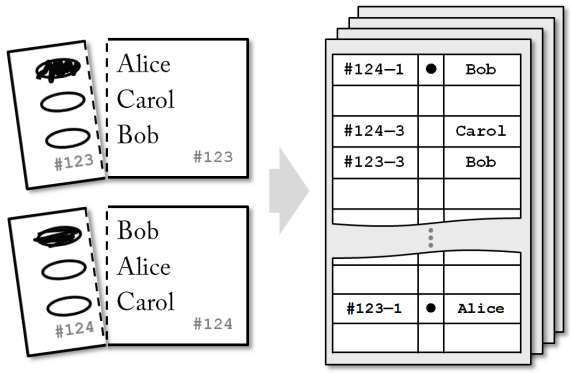


Figure 7.1: **Left: E2E-enabled optical scan ballots.** Each ballot consists of a unique serial number, a candidate list printed in an independent random order, and a perforation between the optical scan ovals and the candidate list. Upon marking the ballot, the candidate list is detached and shredded. The remaining piece is scanned and granted as a receipt. Because the candidate orderings are independent across ballots, knowing the mark position alone does not reveal how a voter voted. **Right: Eperio table.** Each optical scan oval (referenced by a serial number and absolute position), its mark-state (marked or unmarked) and the corresponding candidate name are recorded in a randomly assigned row.

elements $w_1, \ldots, w_u$ in permuted order. For some s which divides u, PermuteBlock applies Permute independently to each of the u/s non-overlapping sub lists in $\mathbf{W}_i$.

Commitment

- $c \leftarrow \mathsf{Commit}(m, r)$
- $\{0, 1\} \leftarrow \mathsf{Reveal}(c, r, m)$

Commit takes as input an arbitrary-length message m and a random factor r. It outputs a commitment to the message c. Reveal accepts values m, r, and c, and outputs 1 iff c is a valid commitment to m and r. Otherwise it outputs 0.

Public Coin Toss

- $\{0, 1\} \leftarrow \mathsf{PublicCoin}()$

PublicCoin returns a uniformly random bit. The output should be unpredictable prior to being invoked and verifiable *a posteriori*.

7.3.4 Lists and Tables

The Eperio protocol relies on a particular data structure, called the Eperio table, which is constructed from a set of private inputs by the trustees. The Eperio table is a novel data

structure and the primary contribution of this chapter. It is shown in Figure 7.1 with a permutation-style ballot, which we use to illustrate the protocol.

To facilitate clarity, we also denote some intermediate lists and tables used in its construction. We use a bold typeface to denote ordered lists and tables, and a script typeface to denote unordered sets.

The following list and set are public inputs to the system decided on prior to the election.

UMR List: $\mathbf{U}$ is the list of each *unique markable region* (UMR) for the s candidates on each of the b ballots. Elements are encoded as a ballot serial number and a position, and are listed in ascending order. The length of the list is $u = s \cdot b$.

Candidate/Selection Roster: $\mathbb{S}$ is the set of selections or candidates to appear on the ballot, for each contest. Elements are encoded as a character string of arbitrary length. The size of the set is s. Without loss of generality, we assume a single contest.

These are used by the trustees, in conjunction with the functions defined above, to create the following private list.

Candidate/Selection List: $\mathbf{S} \leftarrow \mathbf{U} \times \mathbb{S}$ is the list of candidates for each position on a ballot composed by randomly selecting, without replacement per ballot, an element from $\mathbb{S}$. It is ordered by $\mathbf{U}$. The length of the list is u.

The marks list will denote the final status of a markable region. It is empty prior to the election and is provided as a public input to the system after every ballot has been cast.

Marks List: $\mathbf{M}$ is the list of marks corresponding to each markable region in $\mathbf{U}$. Elements include marked (1), unmarked (0), and print audited (-1). The length of the list is u.

The concatenation of these three lists defines a table that collects all the private information of the election.

Print Table: $\mathbf{P}$ is a table formed by joining $\mathbf{U}, \mathbf{M}$, and $\mathbf{S}$. The dimensions of the table are $u \times 3$.

A proof of election integrity subsumes a proof that the relations between each list in $\mathbf{P}$ is consistent with a universal view of the election. Pairwise, a correct **U-M** relation implies ballots were collected as marked (collected-as-marked), a correct **U-S** relation implies ballots were printed correctly (marked-as-intended), and a correct **M-S** relation implies the ballots were counted as collected (counted-as-collected). Note that revealing $\mathbf{P}_{(i,j)}$ for all i and j is sufficient, under our assumptions, for independently verifying the correctness of the tally. Unfortunately, this trivial approach would also destroy the privacy preserving property of the ballot receipt—in conjunction with $\mathbf{P}$, receipts would provide proof of which candidate was selected. Instead, we require a non-trivial approach that can both establish integrity and preserve ballot secrecy.

The Eperio table is a data structure that, with a set of queries, can prove the correct formation of $\mathbf{P}$ while maintaining the same level of privacy provided by only revealing the list of receipts and the final tally. Specifically, it is a collection of x instances of $\mathbf{P}$ that have been independently shuffled row-wise. By revealing portions of and relations on this structure, we will show that a complete and robust proof of integrity can be established with this minimal disclosure.

Eperio Table: $\mathbf{E}$ is a table formed by x instances of $\mathbf{P}$, each of them independently shuffled row-wise. The dimensions of the table are $u \times 3 \times x$.

7.3.5 Protocol

We now outline the protocol for generating an Eperio table and the various outputs required for proving it encodes a correct tally. The focus of this chapter is on the protocol for verifying this proof, which is orthogonal to the issue of how the data is generated. However the security proof we provide in the next section encompasses the generation of the data, and so we give it consideration. The first three steps of the protocol are conducted prior to the election: initial setup, generating the Eperio table, and generating the commitments to the Eperio table. These steps are performed with a *blackbox computation* (for more on this primitive, see Section 7.5).

Initial Setup

The setup assumes that a list of candidates, $\mathbb{S}$, is available as well as the number of ballots, b, to be used in the election. The first task is for the trustees to generate an election secret and receive shares of this secret.

- $(y_1, \ldots, y_n, \kappa) \leftarrow \mathsf{DKG}(n, t, l, f_1, \ldots, f_n)$

Generate Eperio table

Next, the trustees generate the Eperio table $\mathbf{E}$. $\mathbf{U}$ is formed by listing the ballot and position numbers in order. To form $\mathbf{S}$, the candidate list is repeated b times and then randomly permuted on a ballot-by-ballot basis.

- $\Pi_S \leftarrow \mathsf{PRNG}(\kappa)$
- $\mathbf{S}_i \leftarrow \mathsf{PermuteBlock}(\mathbb{S}^b, s, \Pi_S)$

Table $\mathbf{P}$ is created by placing $\mathbf{U}$, $\mathbf{M}$, and $\mathbf{S}$ beside each other in columns. $\mathbf{M}$ is initially empty, but in future meetings will include the marks recorded during the election. $\mathbf{P}$ is used to print the ballots. We use the symbol : to denote an entire vector within a matrix.

- $\mathbf{P}_{(:,1)} \leftarrow \mathbf{U}$
- $\mathbf{P}_{(:,2)} \leftarrow \mathbf{M}$
- $\mathbf{P}_{(:,3)} \leftarrow \mathbf{S}$

Finally, x independent row-wise shufflings of $\mathbf{P}$ are generated. Each shuffled instance of $\mathbf{P}$ is stored in the Eperio table, $\mathbf{E}_{(i,j,k)}$.

- For $k = 1$ to x,
  - $\Pi_E \leftarrow \mathsf{PRNG}(\kappa)$
  - $\mathbf{E}_{(:,1,k)} \leftarrow \mathsf{Permute}(\mathbf{P}_{(:,1)}, \Pi_E)$
  - $\mathbf{E}_{(:,2,k)} \leftarrow \mathsf{Permute}(\mathbf{P}_{(:,2)}, \Pi_E)$
  - $\mathbf{E}_{(:,3,k)} \leftarrow \mathsf{Permute}(\mathbf{P}_{(:,3)}, \Pi_E)$

We define, for future use, the following function. It encapsulates all the steps in this 'generate Eperio table' section.

- $\mathbf{E} \leftarrow \mathsf{GenE}(\kappa, x)$

Generate Commitments

The trustees are now ready to commit to the data in $\mathbf{E}$. For each instance $1 \leq k \leq x$, they will commit to both the $\mathbf{E}_{(:,1,k)}$ and $\mathbf{E}_{(:,3,k)}$ columns. This requires $x \times 2$ commitments. The random factors for these commitments are stored in an $x \times 2$ table $\mathbf{R}$. The resulting commitment is stored in the corresponding table $\mathbf{C}$.

- For $i = 1$ to 2, $j = 2i - 1$, and $k = 1$ to x,
  - $\mathbf{R}_{(i,k)} \leftarrow \mathsf{PRNG}(\kappa)$
  - $\mathbf{C}_{(i,k)} \leftarrow \mathsf{Commit}(\mathbf{E}_{(:,j,k)}, \mathbf{R}_{(i,k)})$

- `Publish` $\mathbf{C}$

Note that $i \to 2i - 1$ is simply the mapping $\{1 \to 1, 2 \to 3\}$, used to denote that commitments to the first and third columns of $\mathbf{E}$ are stored, respectively, in the first and second rows of $\mathbf{C}$. $\mathbf{C}$ is published to the bulletin board.

Voting

Registered and authenticated voters are issued a paper ballot with a randomized candidate list according to $\mathbf{P}$. After marking the ballot, the candidate list is detached and destroyed (*e.g.,* placed in to a paper shredder). The remaining strip is scanned by an optical scanner and the strip is retained by the voter as a receipt. The optical scanners will record for each ballot which position was marked, as well as the ballots that were print audited. After the election, these are placed into the list $\mathbf{M}$. Without knowing $\mathbf{P}_{(:,3)}$, this information does not reveal which candidate was voted for and can be published.

- `Publish` $\mathbf{M}$

Compute Tally

At least $t + 1$ trustees submit their election secrets to the blackbox computation, which regenerates the key and the Eperio table. This time, the completed marks list is shuffled along with the rest of the table.

- $\kappa \leftarrow \mathsf{KeyRec}(y_1, \ldots, y_{t+1})$
- $\mathbf{E} \leftarrow \mathsf{GenE}(\kappa, x)$

For each instance x, they publish the corresponding marks list. Each of these lists is a shuffled version of the original $\mathbf{M}$.

- `Publish:` $\mathbf{E}_{(:,2,:)}$

Finally, a tally is computed from any $\mathbf{E}_{(:,2,k)}$ and $\mathbf{E}_{(:,3,k)}$ pair of columns, and the list of totaled values for each candidate, denoted τ, is published. This can be considered an *asserted tally*, as the purpose of E2E verification is to prove that this tally is correct.

Generate the Linkage List

To ensure that the Eperio table is consistent with what actually appears on the printed ballots in the election, verifiers have the ability to keep an issued ballot for purposes of

auditing its printing. If a ballot was chosen to be print audited and the first position contained candidate Bob, then a row corresponding to this markable region will exist in $\mathbf{E}$ at some row. The row will be different for each instance. Suppose it is in row a in instance k. If the ballot is printed correctly, $\mathbf{E}_{(a,1,k)}$ should contain the correct serial number and position, while $\mathbf{E}_{(a,3,k)}$ should contain Bob. The print auditor would like assurance of this fact.

The problem is that only commitments to entire columns $\mathbf{E}_{(:,1,k)}$ and $\mathbf{E}_{(:,3,k)}$ exist. This prevents us from revealing individual rows like $\mathbf{E}_{(a,:,k)}$. We cannot reveal both $\mathbf{E}_{(:,1,k)}$ and $\mathbf{E}_{(:,3,k)}$ for a given k, as this would break ballot secrecy. Instead, the election trustees will indirectly establish that the row is correct. The trustees assert the row number, a, in each instance corresponding to every audited markable region. Which markable regions are audited is contained in the marks list, $\mathbf{M}$, with -1 recorded for that entry. The list of asserted row numbers is called the linkage list, $\mathbf{L}$, and it is made public.

- for $i = 1$ to u and $k = 1$ to x:
  - if $\mathbf{M}_i$ = -1:
    - `Find:` a s.t. $\mathbf{E}_{(a,1,k)} = \mathbf{U}_i$
    - $\mathbf{L}_{(i,k)} \leftarrow a$
- `Publish` $\mathbf{L}$

Audit Challenge and Response

After the tally has been posted, the trustees prove to an independent auditor that the tally was calculated correctly. They do this through a cut-and-choose protocol. First the trustees regenerate the election secret and the Eperio table.

- $\kappa \leftarrow \mathsf{KeyRec}(y_1, \ldots, y_{t+1})$
- $\mathbf{E} \leftarrow \mathsf{GenE}(\kappa, x)$

Next, they invoke the public coin toss function to generate one flip for each of the x instances.

- for $k = 1$ to x:
  - $z \leftarrow \mathsf{PublicCoin}()$
  - $\mathbf{Z}_k \leftarrow z$
  - `Publish:` $\mathbf{R}_{(z+1,k)}$
  - `Publish:` $\mathbf{E}_{(:,2z+1,k)}$

Each flip is recorded in $\mathbf{Z}_k$. Depending on the flip, they either reveal the first two or

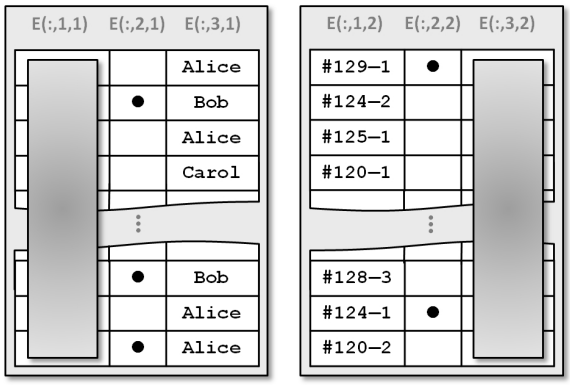


Figure 7.2: **Auditing Eperio table instances**. Two example instances of the Eperio table during auditing. Each instance alleges to contain the same information, but in an independently shuffled order. **Left:** `E(:,3,1)` was challenged (then revealed), allowing verifiers to tally the election. **Right:** `E(:,1,2)` was challenged (then revealed) allowing voters to check their receipts. The grey bars symbolize cryptographic commitments that will remain unopened to protect ballot secrecy.

last two columns in each instance. Recall that $\mathbf{E}_{(:,2,:)}$ was published previously. This is illustrated in Figure 7.2.

7.3.6 Verification

We now show the steps that the verifiers take to check that the published data corresponds to a tally that is correct. Recall there are three sets of audits (and corresponding verifiers). The first set, $\mathcal{V}_1$, are the voters who check their receipts (or provide a copy to someone they delegate to check on their behalf). If the receipt corresponds to ballot number b and contains s positions that are either marked (1) or unmarked (0), the auditor should check that the status of each position i on the receipt matches the status recorded at $\mathbf{M}_j$, where $j = s(b-1)+i$.

The second set of verifiers, $\mathcal{V}_2$, should check the linkage list against their print audited ballots. Let $a = \mathbf{L}_{(i,k)}$ for an i on their ballot and an instance k. Depending on the random coin for instance k, $\mathcal{V}_2$ should check that $\mathbf{E}_{(a,1,k)}$ matches the associated markable region on the ballot or $\mathbf{E}_{(a,3,k)}$ matches the associated candidate. They should do this for all i on their print audited ballot and each k in the election.

These two audits establish that the reported marks correspond to what appeared on voter's completed ballots and that what appears on the ballot corresponds to what is in the pre-committed Eperio table. The final step is to ensure that the asserted tally, τ, corresponds to the marks. Recall that for each of the x instances, a random coin was flipped to reveal value $z \in_r \{0,1\}$. The third set of verifiers, $\mathcal{V}_3$, should do the following.

- for $k = 1$ to x:
  - $z \leftarrow \mathbf{Z}_k$
  - $\texttt{Check}\ \mathsf{Reveal}(\mathbf{C}_{(z+1,k)}, \mathbf{R}_{(z+1,k)}, \mathbf{E}_{(:,2z+1,k)})$
  - If $z = 0$:
    - $\texttt{Check}\ \{\mathbf{E}_{(:,1,k)}, \mathbf{E}_{(:,2,k)}\} \cong \{\mathbf{U}_i, \mathbf{M}_i\}$
  - If $z = 1$:
    - $\texttt{Check}\ \{\mathbf{E}_{(:,2,k)}, \mathbf{E}_{(:,3,k)}\} \cong \tau$

Here $\cong$ means that the two pairs of tables are isomorphic—*i.e.,* they are a permuted representation of the same information.

7.4 Security Analysis (Abstract)

In this section, we summarize the main results of our security proof, which can be found in Appendix C. Given a transcript of the entire protocol, the asserted tally can be either accepted or rejected. If the transcript is correct and matches the asserted tally, the decision will always be to accept (completeness). If the asserted tally is not correct, the decision will be to reject with a high probability (soundness). Finally, the outputs do not provide any information that can be used by a computationally bounded adversary to determine any non-negligible information about which candidate was voted for by any voter (computational secrecy).

Let $\mathcal{P}$ be an unbounded prover (the election authority) and $\mathcal{V}$ be a PPT-bounded verifier. Either entity may employ a malicious strategy and we denote this with a prime $(\mathcal{P}', \mathcal{V}')$. Recall that τ represents the asserted tally and let ρ be the asserted receipts.

Soundness. The soundness of Eperio relies on two assumptions:

1. The function $\mathsf{Commit}(m, r)$ is perfectly binding. That is, for any m_1 such that $\mathsf{Commit}(m_1, r_1) = c_1$, there does not exist any r_2 and $m_2 \neq m_1$ such that $\mathsf{Reveal}(c_1, r_2, m_2) = 1$.
2. The function $\mathsf{PublicCoin}()$ is perfectly unpredictable before invocation.

Let b'_r be the number of modified ballot receipts, and $0 \leq p_1 \leq 1$ represent the fraction of voters who conduct a receipt check. Let b'_p be the number of misprinted ballots, and $0 \leq p_2 \leq 1$ be the fraction of ballots that are print audited. Recall there are x instances of $\mathbf{E}_{(:,:,k)}$, for $1 \leq k \leq x$. Given the above assumptions hold, it is proven (in Appendix C) that the probability of $\mathcal{V}$ rejecting a malformed transcript from $\mathcal{P}'$ is:

$$\mathbf{Pr}[\texttt{REJECT}_{\mathcal{P}',\mathcal{V}}] = \min[(1-(1-p_1)^{b'_r}), (1-(1-p_2)^{b'_p})(1-\frac{1}{2^x})].$$

Computational Secrecy. The secrecy of Eperio relies on the following assumptions:

1. The maximum number of colluding trustees is t.
2. At least one trustee submits to DKG an f_i drawn with uniform randomness from $\mathbb{Z}_q^t$.
3. All outputs are computed with a blackbox.
4. Any polynomial-sized output from $\mathsf{PRNG}(\kappa)$ provides no non-negligible advantage to a PPT-bounded adversary in guessing either κ, the next bit in the output, or any unobserved previous bit.
5. The function $\mathsf{Commit}(m, r)$ is semantically secure and computationally hiding. That is, given either $c_1 = \mathsf{Commit}(m_1, r_1)$ or $c_2 = \mathsf{Commit}(m_2, r_2)$ for any chosen m_1 and m_2 of the same length, a PPT-bounded adversary should have no non-negligible advantage in guessing which message was committed to.

Let ϵ_{A4} and ϵ_{A5} be the advantage specified in assumptions 4 and 5. Given all of the assumptions hold, we prove that the advantage of a malicious verifier, $\mathcal{V}'$, recovering non-negligible information about any voter's selection ($\mathsf{RecoverSel}$) given a view ($\texttt{View}_{\mathcal{V}'}(\cdot)$) of the full transcript as opposed to just the final tally is:

$$\begin{aligned}|\mathbf{Pr}[\mathsf{RecoverSel}(\texttt{View}_{\mathcal{V}'}(\rho, \tau, \mathbf{C}, \mathbf{L}, \mathbf{E}_{(:,2,:)}, \mathbf{R}_z) = 1] - \\ \mathbf{Pr}[\mathsf{RecoverSel}(\texttt{View}_{\mathcal{V}'}(\tau)) = 1]| \leq \\ \epsilon_{A4} + \epsilon_{A5}.\end{aligned}$$

Additional Claims. In addition to the above proofs, we also show that Eperio is complete and can be modified to have everlasting privacy (*i.e.,* $\mathcal{V}'$ is unbounded and $\mathcal{P}$ is PPT-bounded).

7.5 Practical Primitives

In this section, we revisit a few of the cryptographic primitives needed in Eperio. In particular, we are interested in options that allow for useful deployment options.

Blackbox Computation. The Eperio protocol requires the generation of the Eperio table to be done using a blackbox computation. While in theory, the task performed by the blackbox could be made into a multiparty computation (where only privacy is required as correctness is provided by Eperio), we instead propose the use of a semi-trusted computer. It is semi-trusted in the sense of only providing *private* evaluation of functions; the *correctness* can be determined through the audit. That said, mechanisms are provided to encourage correct evaluation. To this end, we assume disclosed source code for the functions to be evaluated is provided in advance and some attestation mechanism is available to ensure it is the same code running on the computer.

All tasks performed by the trustees can be accomplished by regenerating the Eperio table, and the regeneration of this table can be accomplished through a threshold of secret shares from the trustees. Therefore the computer should not have any persistent memory and its internal state should be purged after the outputs have been published. While this assumption may seem strong, in each of the recent occasions where end-to-end verifiable voting systems were used in real-life binding elections, semi-trusted computers were deployed: *e.g.,* Punchscan at the University of Ottawa [ECCP07a], Helios at Université Catholique de Louvain (for key generation from the description in their paper) [AMPQ09], and Scantegrity at Takoma Park, MD [CCC+08]. Unlike Punchscan and Scantegrity, the design of Helios allows easy distribution of the computation to ensure privacy, but a single entity was used for practicality in this case.

Cryptographic Commitment. We are interested in using symmetric-key file encryption as a commitment function for its speed, simplicity and widespread availability of software implementations. For Eperio, this means putting the columns of the Eperio table into individual files, encrypting them under a randomly chosen key, and posting the encryption as a commitment. To open the commitment, the encryption key is revealed and the file can be decrypted. While our implementation of Eperio can be easily modified to work with any standard commitment function, we use this approach to simplify the experience for voters who want to verify the proof for themselves. File encryption utilities are readily available, easy to use, and the commitment has message recovery.

Let $\mathcal{E}$ be a pseudorandom permutation (PRP): $\{0,1\}^k \times \{0,1\}^m \rightarrow \{0,1\}^m$. Let M be a message of L m-bit blocks and let IV be a random m-bit initialization vector. Define E to be the cipher block chaining (CBC) mode of encryption that encrypts $M = m_1, \ldots, m_L$ under k-bit key K and m-bit IV. E is defined as: $c_0 = IV$ and $c_i = \mathcal{E}(K, c_{i-1} \oplus\ m_i)$. Assume M is exactly $L \cdot\ m$ bits long and $k = m$. Let D be its inverse decryption function applied to $C = c_0, \ldots, c_L$ under K: $M = D(K, C)$.

Theorem 1: As defined, E is indistinguishable under a chosen plaintext attack (CPA).[BDJR97]

Conjecture 2: As defined, D behaves like a pseudorandom function with respect to collisions when C is held constant (note the difference from standard assumptions on M and C with a fixed K). That is, $M \leftarrow D_C(K)$ has random collisions for a fixed C and a variable K.

Theorem 3: Define a *commitment with message recovery* function as $(c, \text{IV}) \leftarrow \mathsf{Commit}(M, K) = E(K, \text{IV}, M \| f(M))$, where E is, as defined, CBC mode with a pseudorandom permutation, and $f(M) = M \| M$ is a redundancy function.[4] Define $M' \leftarrow \mathsf{Reveal}(C, K) = D(K, C)$. M' is only accepted when M' has the correct form $M \| f(M)$ for some M. We show that such a commitment is computationally hiding under Theorem 1 and statistically binding under Conjecture 2.

We prove Theorem 3 in Appendix C. Because the commitment has message recovery, its Reveal function differs slightly from the commitment used earlier. We have demonstrated a very specific statistically hiding commitment function that can be constructed from a block cipher, assuming conjecture 2 holds. We model this ideal functionality in the real world with AES-128-CBC in the next section.

Public Coin Toss. Voting systems often require the use of a public coin for the purposes of fairly implementing the cut-and-choose aspect of the audits. In the case of conventional voting, it is used to select precincts for manual recounts. Cordero *et al.* suggest a protocol using dice [CWD06]. Clark *et al.* note that dice outcomes are only observable by those in the room, and suggest a protocol for auditing E2E ballots using stock market prices [CEA07], which has recently been given a more formal analysis (Chapter 6). A further alternative is to use the Fiat-Shamir heuristic [FS86], which is secure in the random oracle model. However, a requirement for Fiat-Shamir is that the challenge space is large. In our case, the number of challenge bits is the same as the number of proof instances—for 10 or 20 instances, Fiat-Shamir proofs can be simulated in the real-world. Thus, we use the stock market protocol we have detailed in Chapter 6.

7.6 Implementation (Abstract)

While the security proofs were abstracted for brevity, we also abstract implementation details from this presentation of the work since the present author contributed very little to these results. For the full implementation details, see the conference paper or its technical report [ECHA10].

[4] We thank and acknowledge Ronald L. Rivest for suggesting this redundancy function for creating a binding commitment from a block cipher. Any errors in the analysis are our own.

Election Generation Tool. A diskless, stand-alone computer is booted from a Linux live CD. The Eperio election generation software is then loaded via a USB key. Both the OS and the software should be attested by the trustees. Trustees enter their passphrases to generate the shared threshold seed and the election data is generated in encrypted CSV files. The public output is written onto the USB stick and the internal state is eliminated by powering the computer down. The election generation software itself is separated into a 'wizard' guided graphical user-interface and back-end components. It was implemented in Python using the GTK+ GUI library, which is a standard to most GNOME Linux live-CD distributions. The primary cryptographic operations were realized using function calls to OpenSSL, also standard. Unlike the Punchscan/Scantegrity software implementation, which requires some initial system configuration, the Eperio trustee interface software is intended to be self-contained and runnable directly following a live-CD boot.

Verification using Code. As a primary objective of Eperio, the verification script was designed to be compact and execute swiftly. A Python script externally calls OpenSSL to decrypt relevant commitment files and then performs the audits on them. We implemented proofs of the Marked-as-Intended and Counted-as-Collected properties in a compact fifty (50) lines of Python code. This represents the smallest implementation of a verification interface relative to other major released implementations by an order of magnitude. We tested this implementation on Ubuntu 7.10 (Python 2.5) and 9.10 (Python 2.6) as well as on Mac OSX Leopard and found that it could be executed without additional installation or configuration. Verifiers using Windows would be required to install Python and OpenSSL, or as an alternative, they could be directed to burn and boot a Linux live CD. With the verification scripts on a USB key, the entire audit can be completed without actually installing or configuring software on a verifier's machine.

Verification using Spreadsheets. An interesting alternative to writing a custom code-base is to use a spreadsheet that can import a CSV file as a worksheet. We believe that spreadsheets can help broaden the appeal of E2E verification, given many citizens who are not accustomed to reading/writing/running code do use spreadsheets. Although many spreadsheets support basic file encryption, cases of improper implementation (as in earlier versions of Microsoft Excel [Wu05]) have led to a general distrust of spreadsheets as an encryption service. We observed that OpenOffice's file encryption leaks partial information about a file's contents by storing an unencrypted/unsalted hash of the first 1024 bytes of the compressed spreadsheet [5], making it unsuitable as an implementation of a hiding commitment scheme (given a decrypted Eperio table, an adversary could test generate other potential tables and test if they match the digest of an encrypted table).

[5]Open Document Format for Office Applications (OpenDocument) v1.2. This is included to provide the user with an error if an incorrect passphrase or key is supplied.

Election	Scantegrity	Punchscan	Helios	Eperio
TAKOMA09	1127s	–	–	162s
GSAED07	–	75s	–	9s
HELv1REF	–	–	14400s	1s
LOC (approx)	1000	2000	1500	50

Table 7.1: Time to verify election audit dataset. Eperio times are for simulated datasets of equivalently sized election. We also include the lines of code (LOC) for each for reference.

In any event, assuming the user can decrypt the revealed tables (whether through the spreadsheet itself or another file-based encryption service like TrueCrypt), the election can be manually verified. The unencrypted CSV file is loaded directly in to the spreadsheet program as a worksheet. The verifier can then complete the audit using only simple spreadsheet operations (*e.g.,* `COPY`, `PASTE`, `SORT`, *etc.*). As an alternative to manual verification, most spreadsheet applications integrate power macro/scripting languages that could automate the audit checks. We implemented a proof of concept macro in Microsoft Excel and tested manual auditing in OpenOffice Calc.

Performance Comparison. We compare our Python verification tool for Eperio to three implemented systems that have been deployed in binding elections: Punchscan, Helios and Scantegrity. Basic timing analysis of election verification on available data is presented in Table 7.1. Our test platform was a 1.8GHz dual-core HP laptop running Ubuntu Linux 9.10. Eperio timings were performed on simulated data for three election datasets: the Takoma Park Scantegrity election (denoted `TAKOMA09`), the GSAED Punchscan election (denoted `GSAED07`), and a reference election for Helios v1. [Adi08] (denoted `HELv1REF`). These were compared to available auditing tools for Scantegrity, Punchscan, and Helios written in Python, C#, and Python respectively. The election dataset for the 2009 election at Universite Catholique de Louvain using Helios v2 was not available [AMPQ09]. This comparison of execution times for verification scripts on three election datasets shows Eperio is significantly faster in time-to-verify.

7.7 Concluding Remarks

The contemporary verification process for electronic voting is deficient. Independent security reviews are generally rare, time-constrained affairs subject to non-disclosure. While we advocate greater transparency for existing election technology, we also contend that end-to-end verification offers a distinct advantage: verification becomes a task of checking election *data* not software and equipment. This may appear to simply shift the problem:

the prevailing methods for verifying E2E election data is with software. However it is not a simple shift. In nearly all E2E systems, the verification code is smaller than the hundreds of thousands of lines of code in a modern DRE. With Eperio, the software is *much* smaller—four orders of magnitude smaller—and verification can even be performed manually without any custom software. By making verification more accessible to voters, we contend that Eperio is an important democracy enhancing technology.

Future Work. Eperio relies on a blackbox computation for privacy. Future work could explore replacing this component with a distributed multi-party computation. Since correctness is handled externally, it may even be possible to replace the blackbox with a distributed protocol in the honest-but-curious model, which would significantly reduce the complexity of the subprotocol. Another avenue for further exploration is alternative constructions of commitments based on block ciphers, and resting the security on standard conjectures or in a suitable model (*e.g.,* the ideal cipher model).

Chapter 8

Toward Secure Printing with Untrustworthy Printers

This chapter references and excerpts from portions of published work from co-authored with Aleks Essex, and supervised by Urs Hengartner and Carlisle Adams [ECHA09]. The current presentation places more emphasis on personal contributions to the project, although each aspect of the project is to some extent joint work.

8.1 Introductory Remarks

In this chapter, we consider how to print a human-readable secret on paper without any individual printer learning anything about what was printed. To accomplish this, we propose a two-party protocol that two printers can use to distributively generate and print secrets. The methods we propose have applications to end-to-end verifiable election (E2E) systems (the use of invisible ink here makes Scantegrity particular amenable), where the complete ballot should not be seen by any individual party except the voter. We present our ideas as a general solution, and leave integration with a specific E2E system for future work.

Contributions. The contributions of this chapter can be summarized as:

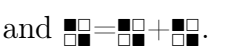
- A scheme for printing arbitrary-length messages using a trusted dealer,
- A scheme for two non-colluding printers to randomly and obliviously select and print a single element from a set of elements,

- A schemes for two non-colluding printers to randomly and obliviously select and print a random permutation on a set of elements,
- An optimization for printing alphanumeric characters using 16-segment display logic.

8.2 Preliminaries

Visual Cryptography. Visual cryptography (VC) is a specialized secret sharing scheme due to Naor and Shamir, where a secret image is split into a number of shares and each share is printed on a transparency [NS94]. When the shares are recombined, by layering them overtop of each other, the secret is restored. There can be any number of shares, and the threshold of shares required to recover the secret can be set at any appropriate value. Work on VC related to this chapter includes the perceptual effects of misaligned shares [LWL09], the use of seven-segment displays with VC [Bor07], and the application of VC to electronic voting [Cha04].

Although the schemes presented here could be distributed across an arbitrary number of printers, for simplicity we consider two-party protocols only. The basic form of VC uses two shares. Consider a secret image s as an $m \times n$ matrix of pixels, where each pixel is either 0 for transparent or 1 for opaque. The first share α is generated by randomly selecting a 0 or 1 for each pixel. This share is then `XOR`ed with the secret image to generate the second share β. Thus, the original can be reconstructed by $s = \alpha \oplus \beta$. However, printing α and β on their own sheet of transparency paper and stacking them is equivalent to an `OR` operation, not an `XOR`.

To correct for this, the basic VC scheme maps each pixel in α and β into a 2×2 block of sub-pixels, which we call a VC-pixel. The mapping is defined as □→  and ■→. By layering VC pixels, we get either a fully opaque VC-pixel, defined as a 1, or a half-transparent VC-pixel, defined as a 0. This emulates the exclusive-or operation where:  , while and .

Invisible Ink. The term "invisible ink" is used to describe a class of inks that, in their initial state, are transparent and un-pigmented, but can become either (i) visible under a certain spectrum of light or (ii) permanently pigmented through a chemical reaction. For invisible inks of type ii, chemical activation of invisible ink can be achieved by (ii-a) moderately heating the paper that the ink is printed on or (ii-b) by applying a liquid chemical to the area containing the ink. For this chapter, we only consider invisible inks of type ii-b.

Recall the Scantegrity system in Chapter 4. There, we deployed invisible inks on optical-scan ballots. The inks were specifically designed to be amenable to use in commercial-off-

the-shelf ink-jet printers, and activated by a developer chemical placed into a special-purpose pen. In its most basic form, one of the color channels of a color inkjet printer is replaced with an ink cartridge containing the invisible ink. Monochromatic graphical objects rendered entirely in this color channel, when printed, are produced in invisible ink instead. To improve the indistinguishability of the invisible ink prior to being developed, especially under black light, we printed randomized patterns with ultraviolet-reactive (type-i) inks to effectively camouflage the message when viewed under an assortment of wavelengths of light [CCC+10].

Document Authentication. If a document has been given to someone, and after some time it is returned, document authentication can provide assurance that the returned document is the same as the original document and not a forgery. Document authentication involves extracting a unique set of features of the document that can be tested for at a later time. Ideally, the features are robust to the document being somewhat mishandled. For physical documents, features can include paper color, paper texture, and ink splatter. Clarkson *et al.* proposes a practical scheme based on fuzzy feature extraction from the three dimensional shape of the paper [CWF+09]. This fingerprinting scheme can be implemented using a commodity scanner and is robust against additional ink being printed on the paper, as well as light mishandling of the document. As part of the following schemes we will require either the printers or recipient to be able to perform some form of document authentication.

8.3 Overview

The core of our proposed solution is a visual cryptography scheme in which individual shares are printed by separate, non-colluding, printers in succession onto a single sheet of paper using invisible ink. While the last printer could print its share in non-invisible ink, we assume that each share is printed in invisible ink. This prevents the last printer's share from being learned if the paper is observed by earlier printers after being fully printed.

The single-sheet approach differs from the original VC proposal, which suggests printing shares on overhead transparencies, which can be aligned on top of a non-transparent share, such as a computer monitor or piece of paper, to reveal the secret [NS94]. The advantage of employing a single-sheet approach over that of a multi-sheet approach is threefold:

- **Alignment:** With VC, proper alignment (or registration) is necessary to reconstruct the message. Misalignments less than a sub-pixel reduce the contrast of the message, while greater misalignments render it unreadable [LWL09]. In the single-sheet scheme, the printers assure the correct alignment of the shares instead of the recipient.

- **Usability:** The role of visual cryptography is not central to the recipient recovering the message and, arguably, the recipient could be completely unaware of it.[1]
- **Tamper-evidence:** Physical separation of shares need not be enforced prior to the recipient receiving them. The recipient will receive decisive feedback if the secret was viewed prior to her receiving it.

In addition to any standard cryptographic assumptions we will outline when relevant, our scheme generally relies on the following physical assumptions:

- A message printed in invisible ink is resistant to exposure by any means other than developing the ink.
- A message printed in invisible ink and developed is easily distinguished from an untouched message.
- No sheet of paper can be substituted for an authenticated sheet of paper without detection.

The third assumption prevents a simple attack where the second printer develops the share printed by the first printer, learns the message by combining it with his share, and then reprints both shares in invisible ink on a new sheet of paper. The key to preventing this line of attack is authenticating the paper prior to printing, and verifying its authenticity at some time after the printing process.

8.4 Print an Arbitrary-length Secret Using a Dealer

General Model. A dealer D wants to have an arbitrary-length secret printed on a sheet of paper, intended for a recipient R, by a third party. D instructs two non-colluding entities offering print service, Printer A and Printer B, on how to print an image of a secret without either printer learning any part of the secret.

Motivating Example. Consider the case where a bank, D, wants to distribute credit card numbers and activation codes to a large set of customers, R, through the mail. Due to the volume, the bank must outsource the printing to a printing service but is concerned that these secret numbers may be surreptitiously compromised. Instead, D would like to distribute the trust between two printing services so that both printers would have to be compromised, or collude with each other, to learn the secrets.

[1] The scheme may require the recipient to have a revealing pen. However in the motivating application, Scantegrity [CCC+08, CCC+10], the use of a revealing pen directly substitutes the use of a normal pen when marking a ballot.

Algorithm 5: Printing a secret with a trusted dealer

Dealer D's Private Input: Secret message $s \in [0,1]^{m \times n}$ as $m \times n$ monochrome pixel matrix

Dealer D should:

1. Fingerprint sheet of paper p
2. Choose $\alpha \in_R [0,1]^{m \times n}$
3. Compute $\beta = s \oplus \alpha$
4. Send α and β to Printer A and B respectively

Printer A should:

5. Print α in invisible ink onto p
6. Send p to Printer B

Printer B should:

7. Print β in invisible ink on top of α on p
8. Send p to Dealer D

Dealer D should:

9. Authenticate paper p
10. Send p to recipient R

Solution. Given our physical security assumptions, we propose a simple distributed printing scheme, outlined in algorithm 5, involving a trusted dealer D (the bank) issuing two VC-shares of a secret to two printers: Printer A and Printer B. Printer A prints its share on a piece of paper in invisible ink. This sheet is given to Printer B, which prints its share in invisible ink, directly on top of Printer A's share. The bank keeps an inventory of sheets issued to Printer A and verifies their authenticity after having the sheets returned by Printer B (it could also at this time reveal the invisible ink). Alternatively, the recipient R could use a special developing pen to reveal the combined shares (and hence the secret) upon receiving the paper.

This scenario has certain drawbacks: if the dealer has the ability to authenticate each sheet and, potentially, to reveal the ink, it should have the capability of printing the secret itself. However it will serve as a stepping stone to the schemes without a dealer in the next sections. There Printer A will publish an inventory of the sheets issued to Printer B. Both printers can check the sheets against the inventory at any point in the protocol, and after Printer B has applied its shares, the sheets can be authenticated by the recipient. An alternative is to use an honest-but-curious third party to provide this service. In all cases, the verification could be conducted through a random audit of a small portion of the sheets.

8.5 Print a Randomly Selected Secret without a Dealer

General Model. Two non-colluding entities offering print service, Printer A and Printer B, randomly and obliviously select a message from a public set of possible messages. The pixel representation (*i.e.,* image) of the message is printed on a sheet of paper intended for a recipient R without the use of a dealer. The selected message is unknown to both Printer A and Printer B and will only be known by R. Furthermore, printers A and B can each enforce the randomness of the selection independent of each other.

Motivating Example. Consider the case where Printer A and Printer B want to print a random alpha-numeric character on a sheet of paper to give to R. By doing this successive times, they could print a multi-character string where each character is independently selected at random. A number of election systems with cryptographic end-to-end integrity require ballots to be printed with various codes. One example is Scantegrity [CCC$^+$08, CCC$^+$10], which requires secret and random codes to be printed beside each candidate and already uses invisible ink to hide the value of the codes. Another is ThreeBallot [RS07], which requires a set of statistically unique identifiers to be printed on each ballot. Printers or poll-workers who observe these identifiers, prior to the ballot being cast, threaten ballot secrecy.

Protocol. Let the set of possible messages be S and of order $N = |S|$. Consider, as in the motivating example above, that $S = \{A \ldots Z, 0, \ldots, 9\}$ in which each $s_i \in S$ will be represented as an m by n monochrome pixel matrix that visually expresses it. At a high-level, Printer A will generate a random visual crypto image, α, as her share, and print it onto the paper in invisible ink. She will then, for each message $s_i \in S$, generate the complementary set of shares for Printer B: $\beta_i = \alpha \oplus s_i$, for $1 \leq i \leq N$. She randomly permutes the order of the set, obfuscates each element, and sends the set to Printer B. Printer B then selects $\gamma \in_R [1, N]$ to be deobfuscated, where $\in_R$ denotes the uniform random selection of an element from a set. He then prints β_γ over top of α in invisible ink.

We require three additional properties of this distributed protocol in the absence of a dealer, namely:

i. Printer A should not learn which visual crypto share β_γ was selected by B.
ii. Printer B should not learn the value of any share β_i other than the single share, β_γ, he selected.
iii. Printer B should not know which message s_i corresponds to β_γ.

We utilize a 1-out-of-N oblivious transfer protocol to achieve properties (i) and (ii). s_i is perfectly hidden by Printer A's share α. Thus (iii) holds under the assumption of non-collusion of the printers.

Algorithm 6: Printing a single secret character with oblivious transfer

Public Parameters: Set of alphanumeric characters S, and generators $g, h \in \mathbb{G}_q$

Printer B should:

1 Choose index to select: $\gamma \in_R [1, N]$
2 Choose random secret: $x \in_R \mathbb{Z}_q^*$
3 Commit to private choice: $y = g^x h^\gamma$
4 Send y to Printer A

Printer A should:

5 Perform lines 1, 2, and 5 from Algorithm 5
for $1 \leq i \leq N$ **do**
6 Select i^{th} message from set: $s_i \in S$
7 Compute complimentary share: $\beta_i = \alpha \oplus s_i$
8 Randomly permute index: $\beta_i \rightarrow \beta_{j=\pi(i)}$
9 Choose random secret: $r_j \in_R \mathbb{Z}_q^*$
10 Compute: $c_j = \langle a_j, b_j \rangle = \left\langle g^{r_j}, \beta_j (\frac{y}{h^j})^{r_j} \right\rangle$
11 Send p and set of c_j's, ordered by j, to Printer B

Printer B should:

12 Flip coin $c = \{H, T\}$ with $\Pr[H] = \rho$
if $c = H$ **then**
13 Request α, β_j, and r_j, $\forall j$, from Printer A.
14 If correct, repeat protocol from Line 5.
else
15 Select c_γ
16 Compute: $\beta_\gamma = \frac{b_\gamma}{(a_\gamma)^x}$
17 Print: β_γ on top of α on paper p

Recipient or Printer A should:

18 Authenticate paper p

The full details of the protocol are provided in Algorithm 6. The oblivious transfer sub-protocol is due to Tzeng [Tze04], which we selected for its reusable public parameters and minimal message exchange (2-pass). It is set in the ring of integers modulo a large prime p, with multiplicative subgroup of prime order q. By using a Pedersen commitment in line 3, Printer B's choice of share is perfectly hidden ensuring (i). Property (ii) holds because line 16 for an arbitrary j reduces to $\beta_j h^{r_j(\gamma - j)}$ allowing the recovery of β_j only when $\gamma = j$.[2] Property (iii) holds because $s_i = s_{\pi^{-1}(j)} = \beta_j \oplus \alpha$, and Printer B does not know α or random permutation $\pi()$.

[2]Security remark: If printer B could easily recover β_j for $j \neq \gamma$, then Printer B could easily perform an arbitrary discrete logarithm: either $\log_g(g^{r_j})$ to recover β_j directly or $\log_g(h)$ to find x' such that $g^{x'} h^j = y$. Concerning the latter, it is thus important that g, h are generated by Printer A or through a distributed key generation (DKG) protocol.

Printer A could misconstruct the set of β_j values such that when they are combined with α, they do not each produce a unique character (*e.g.,* any selection by Printer B will result in the same character being printed). Given property (ii), Printer B could not detect such an attack directly. Thus Printer B will, with some probability ρ, perform a cut-and-choose audit of Printer A's construction of α and β_j values to ensure they are properly formed.[3]

An additional feature of the protocol is that both parties contribute to the *random* selection of $s_i \in S$. Printer A chooses a random permutation $\pi : i \rightarrow j$, and Printer B selects a random index γ to print. Thus if one of the two parties select messages deterministically, the contribution of the other will be to ensure the printed message is, in fact, randomly selected.

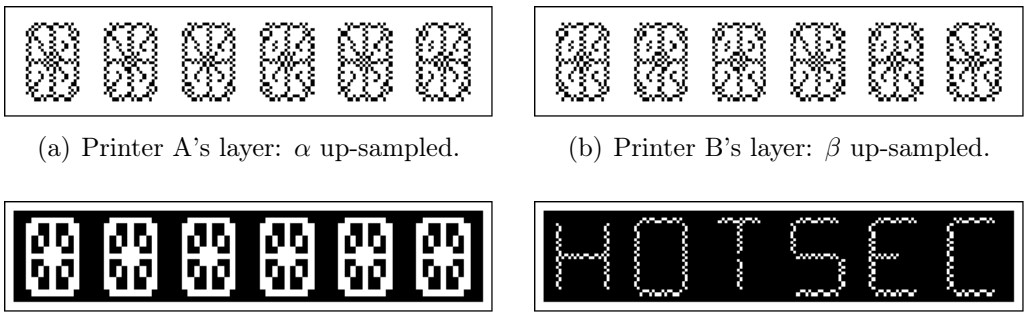
This protocol outlines how to print a single, secret, random, and obliviously selected alphanumeric character on a piece of paper. It is easy to see that the oblivious transfer could be conducted several times, independently, to produce a string of characters for applications such as those offered above as motivating examples.

8.6 Efficient Textual Representation

General Model. Two non-colluding printers, Printer A and Printer B, randomly select a short string of alphanumeric characters from a public set of possible strings. The model has the same properties as the general model in Section 8.5 but is optimized for the use of alphanumeric strings. It can be used in conjunction with Algorithms 6 and 7.

Motivating Example. An official is to issue a survey that contains a question about sensitive information that respondents may not answer honestly for fear of retribution. A mitigating technique is randomized response, where a sensitive question can be replaced with its negation in a random fraction of the surveys [AJL04]. Thus the issuers do not learn any particular respondent's true response with certainty, but can statistically adjust the results to estimate the number of respondents who answered in a particular way. Consider the problem of obliviously printing a survey, where a particular question is randomly selected from a set that contains nine elements of question Q and one instance of question ¬Q. The answers are returned on a different sheet of paper (*e.g.,* a Scantron form), and the question sheet is discarded by the respondent after using it. The mechanism in this section can also be used to optimize the motivating examples in Section 8.7: for printing candidate names in election systems or the names of prizes in a contest.

[3]It may be possible for Printer A to formulate a Σ-protocol proof instead, however the proof of knowledge is complicated by `XOR` relations.



(a) Printer A's layer: α up-sampled.

(b) Printer B's layer: β up-sampled.

(c) Static background mask.

(d) The three layers combined.

Figure 8.1: **Visual Crypto Up-sampling.** Example of a string of text using 16-segment up-sampling, comprised of two visual crypto layers set against a static background. Instead of representing each segment as a collection of traditional 2×2 visual crypto pixels, each segment's share can be transmitted as a single bit and then 'up-sampled' to an arbitrarily large, public, and pre-arranged visual crypto pixel pattern for improved perceptual clarity. In this way visual crypto shares can be fully expressed by 16 bits per alphanumeric character, regardless of perceptual resolution.

Solution. A mechanism for solving this problem has already been proposed: run Algorithm 6 with the questions as full strings, instead of characters. However there may be an upper-limit to the size of the message that can be transferred in one instance of the protocol: in this case, it is the security parameter of the system, which could optimistically be set to 1023 or 2047 bits (*i.e.,* where $p = 2q + 1$). Of course p could be made larger with a custom implementation: in any event, for the purpose of efficiency, our motivation is to encode as many characters as possible into one ciphertext payload.

We defined messages to be a monochrome pixel matrix of dimensions $m \times n$. Consider, for example, the image of a character to be 26 by 18 sub-pixels that are either white or black (the resolution that will be used in Figure 8.1). These parameters would require 117 bits per character, allowing just over half a dozen characters to fit into a single ciphertext payload. Increasing the resolution (and hence the readability) comes at the cost of using additional encryptions to convey the same content. To improve on this, we can use segment displays. In a segment display, alpha-numeric characters are displayed by driving a subset of 16 segments, or 7 segments if we restrict ourselves to numbers. At 16 bits per character, we can fit over 60 alpha-numeric characters into one 1024-bit secret: enough to encode a short question, candidate name—regardless of the character resolution.

With the use of Algorithm 6, Printer A can obliviously transfer one of many message shares to Printer B, where the share (β_γ) is a sequence of 16-bit segment encodings. If the exclusive-or of this sequence is taken with Printer A's accompanying sequence (α), the result is the character encoding of a randomly selected string (s_γ). However, we deviate from Algorithm 6 in that Printer A and B cannot print α and β_γ directly. Instead Printer A and B must up-sample their 16-segment sequences into a VC share.

The process of up-sampling works as follows. A pixel matrix of any dimension is partitioned into segments and background; white and black in Figure 8.1(c) respectively. To aid with perception, the segment portion (white portion) is filled with random-looking VC pixels (recall a VC pixel is a 2x2 sub-pixel representation of a pixel). This same pixel mask can be used for every character. Printer A up-samples α by taking the pixel mask and for every segment, either leaving the segment as is if α is 0 for that segment or flipping all the bits in the segment if α is 1. See Figure 8.1(a). Printer A prints this and the background mask in invisible ink. Similarly Printer B up-samples β using the same pixel masks, generates a VC share in Figure 8.1(b), and prints it.

8.7 Print a Permutation of a Set of Messages without a Dealer

General Model. Two non-colluding entities offering print service, Printer A and Printer B, randomly select a permutation and apply it to a public set of possible messages. The pixel representations (*i.e.,* images) are printed on a sheet of paper intended for a recipient R without the use of a dealer. The order of these images is unknown to both Printer A and Printer B and will only be known by R. Furthermore, printers A and B can each enforce the randomness of the permutation independent of each other.

Motivating Example. A number of election systems with cryptographic end-to-end integrity require ballots to be printed with a randomized candidate ordering. Prêt-à-voter [CRS05] and Aperio [ECA08, ECA10] require candidate names to be listed on the ballot in an independent random order. The candidate list could be developed immediately prior to voting to increase voter privacy. Note that our solution would still require an additional mechanism for setting up the election data consistently with the randomly chosen permutation (see future work in Section 8.8).

Alternatively, consider a contest where the names of prizes are printed in invisible ink on a set of tickets. For example, a batch of one dozen tickets could be printed as follows: ten will say "please play again" while the other two will each name a different prize. By applying a random permutation to these twelve strings, not even those running the contest will know for certain which tickets contain a prize.[4]

Solution. A permutation of a set of N images requires N elements to be printed. In its most basic form the 1-out-of-N oblivious transfer (see algorithm 6) is run N times, with

[4]This could prevent documented cases of fraud, such as `http://archives.cnn.com/2001/LAW/08/21/monopoly.arrests/`

Algorithm 7: Printing a secret permutation with oblivious transfer

Public Parameters: Superincreasing indices $\kappa = \{\kappa_1, \cdots, \kappa_N\}$ that sum to $\hat{\kappa}$.

Printer B should:

1 **for** $1 \leq k \leq N$ **do**
2 Choose, without replacement, index: $\gamma_k \in_R \kappa$
3 Choose random secret: $x_k \in_R \mathbb{Z}_q^*$
4 Commit to private choice: $y_k = g^{x_k} h^{\gamma_k}$
5 Send y_k to Printer A
6 Compute $\hat{x} = \sum_{k=1}^{N} x_k$ and send to Printer A

Printer A should:

7 Compute: $\hat{y} = \prod_{k=1}^{N} y_k$
8 Verify: $\hat{y} = g^{\hat{x}} h^{\hat{\kappa}}$
9 Flip coin $c = \{H, T\}$ with $\Pr[H] = \rho$
10 **if** $c = H$ **then**
11 Request γ_k, x_k, and y_k, $\forall k$, from Printer B.
12 If correct, repeat protocol from Line 1.

13 **for** $1 \leq k \leq N$ **do**
14 Run Algorithm 6 at line 5.

Printer A selecting N independent, random, visual crypto shares α. However instead of applying independently selected permutations $\pi_1, \cdots, \pi_N$ at each successive execution, the same random permutation π_1 is applied to S when constructing the complementary set of VC shares $\{\beta_{(1,k)}, \cdots, \beta_{(N,k)}\}$ during the k-th execution.

However since a permutation of elements requires every element to be appear once and only once, we will require a mechanism to enforce non-repetition of elements. Such non-repetition of VC shares constructed by Printer A can be made through a similar cut-and-choose process as that mentioned in section 8.5. However to enforce non-repetition of the selections made by Printer B, we extend algorithm 6 by algorithm 7 such that Printer B proves the uniqueness of her selections, γ_k, without revealing the order of selection, as well as providing Printer A with the ability to perform a cut-and-choose on Printer B's selections.

As a brief description of algorithm 7, Printer B constructs N commitments $y_k = g^{x_k} h^{\gamma_k}$ and sends them, along with the sum of random factors $\hat{x}$ to Printer A. We index each message in the set using a public set of indices, selected from a superincreasing sequence κ (*e.g.,* $\{k \in \mathbb{Z} : \kappa_k = 2^k\}$). Thus, if Printer B selects the same index more than once, it is impossible to adjust the other selections such that they sum to $\hat{\kappa}$ and are valid indices. Printer B could select an index not in κ, however this would forfeit him from learning at least one proper share. This will be caught by the cut-and-choose subprotocol in lines 9-12,

which can be thought of as an optional step for scenarios where a malicious printer could get away with misprinting a share.[5] To verify that each of the commitments y_k contains a unique index choice, Printer A will calculate their product $\hat{y}$ causing the exponents γ_κ to sum. Given public parameter $\hat{\kappa}$ and Printer B's assertion $\hat{x}$, Printer A will verify whether $\hat{y} = g^{\hat{x}} h^{\hat{\kappa}}$ thus verifying B's honesty in selecting each element once.

8.8 Concluding Remarks

We have demonstrated an interesting, novel paradigm: selected messages can be printed on a sheet of paper without the printers learning them. We have outlined a number of scenarios where such a property may be useful, including password distribution, cryptographic voting, contests, and randomized response surveys. Indeed these protocols may be relevant to other applications of visual cryptography, in particular those requiring a dealer-less solution, as well as to other types of document and display media. We hope this work points to new possibilities now that we know how to print a secret.

Future Work. The integration of the protocols presented here into an E2E election system is an open problem and is non-trivial. For example, in the case of Scantegrity, the printers could obliviously generate and print random confirmation codes for each ballot. However these codes need to also exist in the election data that is committed to prior to the election, for use in print audits, receipt checks, and dispute resolution. For this reason, we entitled the paper *toward* secure printing with untrustworthy printers.

[5] For example, in the voting scenario, the permutation will be observed by the voter to be correct before it is used, thus not requiring this optional step. However it may be desirable for printing tickets in a contest.

Part II

Internet Voting

Chapter 9

Related Work on Internet Voting

9.1 Introductory Remarks

In Part I of this dissertation, we considered methods for achieving integrity, ballot secrecy, and incoercibility in in-person voting protocols, while striving for a high level of deployability. The difference between an E2E system and a hand-counted paper ballot voting system, where an observer diligently watches the entire process, is mainly one of convenience. Voters can cast-and-go and still receive the same level of assurance as the diligent observer for all precincts.

In this second part of the dissertation, we consider developing E2E systems for remote voting. As with booth voting, remote voting has its present day counterparts: vote-by-mail and internet voting. But unlike booth voting, there is a large gap between an E2E remote voting system and either of the baseline systems. Consider standard vote-by-mail: there is no guarantee a vote is included in the tally or that it is unmodified, even to a diligent observer, and voters can fill out their ballots in front of anyone they choose. Mailed-in votes must contain identifiable information to verify it came from an eligible voter. Ballot secrecy could be met if the ballot itself is placed inside a second indistinguishable envelope that is mixed in with other eligible ballots but the voter has no assurance that procedures are followed.

In this section, we consider how to approach remote voting while maintaining the same level of security as E2E in-person election systems. Most of the approaches we review make extensive use of cryptography, and are thus more suitable for internet voting than vote-by-mail. Internet voting has the advantage of allowing interactive protocols but the disadvantage of using what can only be assumed to be an untrustworthy platform: the voter's computer.

9.2 Security Model

The prospect of voting online inherits all the problems of precinct-based voting while introducing new ones. In particular, we highlight three fundamental problems that emerge: (1) voting is conducted on an untrustworthy computer that may be infected with malicious software, (2) the privacy of a voting booth is removed, opening up the possibility of voter coercion and vote buying schemes, and (3) there is an increased susceptibility to denial of service and network attacks.

In one sense, the untrustworthy platform problem is not any different than voting on an untrusted DRE in an E2E system. Recall that these systems generally provide the voter with a receipt that she can use to verify that the DRE followed the protocol. The receipt check is generally done from the voter's computer, which is also untrustworthy, but should not pose a threat as long as the DRE does not collude with the voter's computer. However in internet voting, the DRE is the voter's computer, which eliminates the separation of duties. Instinctively, we might suggest that voters vote from one computer and check from another. If we generalize this idea to the assumption that voters can use more than one independent channel, then most of the solutions to the untrustworthy platform problem fall within this framework.

The second problem, the removal of the voting booth, provides a unique challenge. It is generally believed that incoercibility is impossible to achieve in the standard model with secure channels alone [HS00]. Instead of eliminating the untappable channel, coercion-resistant voting systems tend to reduce the reliance on them using one of two approaches: (a) use multiple secure channels and assume that while any individual channel can be tapped, no adversary can tap all channels simultaneously (*cf.* perfectly secure message transmission [DDWY93]), or (b) use an untappable channel once during a registration phase and bootstrap this interaction into an arbitrary number of future interactions over secure channels.

The third problem, (distributed) denial of service attacks, is typically considered out-of-scope. If any appeal is made, it is to the general techniques for preventing DDOS attacks against any online service. With the exception of the "board-flooding" attack we review below, voting systems are generally no different from other web-services regarding these attacks. As with general web-services, the server collecting the votes also needs to be hardened to prevent access from any online malicious party (whereas tampering with a DRE requires physical access).

Most of the literature concentrates on resolving either the first or second problem. We will review proposals for both but our own contributions will only be toward the second.

9.3 Untrustworthy Platform Problem

Code voting. The first proposal for securely voting from an untrustworthy platform was SureVote by Chaum [Cha01]. Voters receive a sheet out-of-band (*e.g.,* by mail) that contains a serial number and a pair of individually random codes for each candidate (a vote code and an acknowledgement code). The voter casts her vote by entering the serial number and the first of the two codes (vote code) associated with her selection. Since the computer does not know the vote codes on the card, it cannot determine which candidate the voter voted for nor can it reliably change her vote to a different, valid candidate. After receiving the vote code, the voting system will respond with the second code (acknowledgement code) which it computes from the serial number and vote code. This computation requires the trustees to be online. After the election, the mapping between codes and candidates can be shown to be correct without revealing it (it would be straightforward to adapt Eperio to allowing this verification).

Helbach and Schwenk propose using an additional lock-in code [HS07]. The vote code and acknowledgement code work as in SureVote, and the lock-in is submitted when the voter is satisfied that her ballot has been correctly received. Helbach, Schwenk, Schäge extend this scheme, allowing voters to receive many code sheets [HSS08]. To prevent double voting, voters sign their codes with a linkable group signature scheme. However the linking phase is not universally verifiable.

Joaquim and Ribeiro adapt SureVote to use with a trusted smartcard [JR07, JRF10]. Voters still receive a code sheet in the mail like in SureVote but the codes are paired with the smartcard (*e.g.,* the sheet was generated with the smartcard during some registration phase). To vote for a candidate, the voter enters the vote code which is routed by their computer to the smartcard. The smartcard constructs a standard cryptographic ballot using some underlying E2E system, left abstract by the authors, for the candidate corresponding to the code. This is forwarded by the computer to the election authority. The smartcard returns the acknowledgement code to the voter when it is satisfied the ballot was received by the authority. Later Joaquim, Ribeiro, and Ferreira [JRF09] describe how their system can interface with the MarkPledge system (see Section 3.4.2) to perform the protocol between the smartcard and election authority. They also propose a matrix-based code sheet that can be reused in multiple elections [JRF10].

Ryan and Teague also propose an E2E backend for code voting called Pretty Good Democracy (PGD) [RT09b]. In PGD, trustees generate a table where each row corresponds to a code sheet and contains an encryption on each candidate's vote code (for that sheet) in a shuffled order, an encryption of the permutation itself, and an encryption of the acknowledgement code. The rows are distributively generated by a set of trustees using a protocol similar to the one used to creating Prêt à Voter ballots (see Section 3.4.3). Voters submit an encryption of their vote code and the trustees compare it, using a plaintext

equality test, to each encrypted code in the row. When a match is found, the matching position is combined with the encrypted permutation like in Prêt à Voter. The trustees can jointly decrypt the acknowledgement code and return it to the voter.

Heiberg, Lipmaa, and Laenen propose a code-voting scheme with emphasis on deployment [HLL10]. It proposes a useful cryptographic primitive, called proxy oblivious transfer, that allows the election authority to convert an encryption of the voter's selected candidate into an encryption of the corresponding acknowledgement code for that voter and candidate without decrypting the submitted vote. The acknowledgement code can then be decrypted by an independent, non-colluding party and sent to the voter over an out-of-band channel (*e.g.,* SMS).

Removing the out-of-band channel. Oppliger, Schwenk, Löhr propose a codevoting scheme [OSL08] where, instead of the voter receiving her codes out-of-band, she receives them as CAPTCHAs [ABHL03] through her untrusted computer. The computer cannot read the codes, although it could shuffle the order. Popoveniuc resolves this shuffling attack in SpeakUp by overlaying the CAPTCHAs on a photograph [Pop10]. In SpeakUp, voters then submit their vote codes by reading them into the computer's microphone, which also prevents the computer from tampering with the vote as it is transmitted back to the election authority. The authority authenticates the voter with voice-detection and then uses an E2E system to verifiably map the codes to the proper candidates.

Permutations. An early alternative to code voting is to send to the voter, out-of-band, a randomized candidate order. The voter selects the candidate, and the computer encrypts the position. The untrustworthy platform does not know the permutation, however unlike with code-voting, it can reliably change the order and so this must be guarded against. Damgard and Jurik offer the first scheme of this kind, where an expensive multiparty computation is used to undo the permutation under encryption [DJ02]. Groth and Salmonsen improve the efficiency somewhat by using the same permutation for groups of voters instead of individually [GS04]. Kutyłowski and Zagórski use cyclic shifts instead of full permutations and voters are sent a pair of sheets, so that one can be audited for correctness and the other used to vote [KZ07].

Open-audit systems. In Section 3.3.7, we listed a number of implementations of the cryptographic protocols we reviewed in Chapter 3. Many of these are meant as internet voting protocols, however they only address verifiability and not coercion or the untrusted platform. Helios does allow the voter (or anyone) to have their browser construct multiple encryptions of her ballot which she can challenge to be opened and audited [Adi08].

9.4 Coercion-Resistance

Informal definition. To define coercion-resistance, consider the following game [JCJ05, KTV10b]. A voter is selected for coercion. The voter has some intention γ_v. For example, γ_v could be to vote for Alice or a more general strategy like vote for a random candidate or abstain from voting. Within the game, a coin is flipped. If the coin is heads, the voter will comply with the coercer. This means the voter tells the coercer everything she knows and follows his exact instructions. More formally, if the voter is an interactive Turing machine (ITM), the coercer learns her internal state and she only forwards messages to and from the coercer (*i.e.,* employs a dummy strategy). If the flip is tails, the voter will try and deceive the coercer. She will give him fake information and when she is instructed to do something, she will act as if she did it while not necessarily complying. More formally, she only simulates the dummy strategy and may send and receive messages without forwarding them to the coercer.

Now consider playing this game with an ideal system where voters give their votes to a trusted party and the trusted party produces the tally. The trusted party flips the coin. If the flip is heads, the trusted party uses the coercer's vote and if it is tails, it uses γ_v. We can ask, in this ideal system, what is the probability that the coercer can guess whether the voter is complying with him or deceiving him? The answer is not 1/2. For example, if the coercer tells the voter to vote for Alice and the final tally is unanimously for Bob, he can determine she did not comply. Similarly, if we give the coercer a probability distribution for what the tally will be (we do not want the security of our system to depend on him not having access to this information), he could experience non-negligible success in determining if the voter is cooperating or not.

With this in mind, consider playing the game with an actual system. We say the system is coercion-resistant with respect to γ_v[1] if two properties hold: (1) if the flip is tails, the voter can always accomplish γ_v and (2) the coercer cannot guess whether the voter is complying or deceiving him with more (or less) success than he could if he played the same game with the ideal system.

JCJ & Civitas. The first internet voting system to achieve coercion-resistance was proposed by Juels, Catalano and Jacobsson [JCJ05]. It was refined and implemented as Civitas by Clarkson, Chong and Myers [CCM08]. We will refer to them interchangeably.

The first step in JCJ is registration. An eligible voter's name is added to a public roster and each of the n trustees generates a random value σ_i and posts $\mathsf{Enc}(\sigma_i)$ (under a

[1]We consider systems that are coercion-resistant with respect to casting a vote according to any probability distribution across the possible candidates (including no candidate). If γ_v could be *anything*, it could be, *e.g.,* to "vote the same way as the voter before me" which is in violation of other properties we want to achieve.

threshold key they share) to the roster beside the voter. Then each trustee sends the voter σ_i and a designated-verifier proof that σ_i matches $\mathsf{Enc}(\sigma_i)$. The voter computes $\sigma = \prod_i^n \sigma_i$, which she will use as a credential for voting, and $\mathsf{Enc}(\sigma)$ is computed homomorphically on the roster.

Let v be the voter's selection and π_1 be the NIZKP that $\mathsf{Enc}(v)$ encrypts a valid candidate. The voter submits $\langle \mathsf{Enc}(v), \pi_1, \mathsf{Enc}(\sigma), \pi_2 \rangle$, where π_2 is a non-interactive proof of knowledge of σ, over an anonymous channel. If an adversary coerces the voter and demands her credential, she chooses a trustee she trusts not to be colluding with the coercer, prepares a claim that she received $\sigma_i' \neq \sigma_i$ from him, simulates a DV-proof to support this assertion, and computes the resulting fake credential σ' using the fake share. She then supplies the adversary with σ' (and her prepared evidence if he demands it). Finally, if she has not already cast her actual vote with her real σ, she does this.

Tallying the submitted votes is broken into several phases. First, the trustees verify π_1 and π_2 for each submission and agree on which submissions have invalid proofs. These are eliminated. Next the trustees will search for any votes that share the same value σ (*i.e.,* double votes). This is done by running a plaintext equality test between all pairs of $\mathsf{Enc}(\sigma)$ values. If V_0 votes were submitted and $V_1 \leq V_0$ have valid proofs, this step is quadratic in V_1. Duplicates are removed according to some policy (*e.g.,* keep the last submitted vote).

Next the trustees use a verifiable mix network to shuffle the remaining submissions. They also shuffle the ciphertexts on the roster of eligible voters. They now compute the intersection of $\mathsf{Enc}(\sigma)$ values in the submitted votes and on the roster. This is also done with a quadratic number of plaintext equality tests, between each submitted value and each roster value. For submissions that match, $\mathsf{Enc}(v)$ is decrypted and the results tallied.

Remarks on JCJ & Civitas. Civitas addresses the issue of the untappable channel by assuming that there is at least one channel between the voter and a trustee that is not tapped, while JCJ assumes that voters interact with a single registrar over an untappable channel (*e.g.,* they suggest by mail or in-person) and that the registrar does not collude with the adversary. In Civitas, the voter must know which trustee is trustworthy so a distributed registration does not buy much. Furthermore, the use of designated verifier proofs assumes the voter has a public key known to the trustees, and so there is likely an in-person registration of the key hidden underneath the assumption of an existing PKI. We only point this out to justify our own use (in Chapter 11) of an in-person registration.

The use of designated verifier proofs also makes the system susceptible to the same attack Michels and Horster [MH96] propose against Sako and Kilian's use [SK95] of trapdoor commitments (see Section 3.3.3) (also pointed out by Küsters and Truderung against Civitas [KT09])). If the adversary coerces the voter into revealing her private key, the adversary can collude with any of the trustees to simulate the DV-proof that the credential

share matches the encrypted value posted on the roster. The voter cannot use such a credential to cast a ballot (violating the first property of coercion-resistance). In JCJ, this is covered by the assumption that the single registrar does not collude with the adversary. In Civitas, it is covered by an additional assumption that voters cannot be corrupted during registration. A variation of this attack could be that the adversary supplies the voter with a public key that she does not know the corresponding private key for (this is mentioned but not resolved by Jakobsson, Sako and Impagliazzo when introducing DV-proofs [JSI96]). In this case, the DV-proof is transferrable since the voter is not actually the designated verifier (the adversary is).

JCJ variants. Following JCJ, a number of attempts have been made at linearizing the complexity of the tallying phase. Both Smith [Smi05] and Weber, Araújo, and Buchmann [WAB07] propose exponentially blinding (as is done within a plaintext equality test) all the $\mathsf{Enc}(\sigma)$ values (both submitted and on the roster) with the same blinding factor and decrypting the result. However an adversary can break the coercion-resistance of this approach [AFT07, CCM08, AFT10]. In fact, it is the same related ciphertext attack Pfitzmann [Pfi94] proposed against the Park *et al.* mix network [PIK93]. To test if a coerced credential σ is real, the adversary submits a vote with $\mathsf{Enc}(\sigma)$ and with $\mathsf{Enc}(\sigma^t)$ where t is a random "tag" value. Such a relationship between the ciphertexts will be preserved under exponential blinding. The adversary can search the decrypted blinded values for a pair that differs by root t, and see if the first value matches a roster value. Later, Araújo *et al.* show [ARR+10] that related ciphertext attacks can be applied to three additional schemes by Acquisti [Acq04], Schweisgut [Sch06], and Meng [Men07].

SKHS. In work concurrent to our work in Chapter 11, Spycher *et al.* have observed that while exponential blinding is not suitable for determining if a credential is on the roster, it can still be used to eliminate duplicate votes [SKHS11]. In this variant, voters submit an encryption of their position on the roster, $\mathsf{Enc}(R_j)$, along with the values in JCJ. For each roster entry, the trustees generate some fake votes. The trustees then use exponential blinding (on the submitted $\mathsf{Enc}(\sigma)$ values) to eliminate the duplicate votes. They then mix the submissions and decrypt the $\mathsf{Enc}(R_j)$ values. For a given R_j, a number of values will be left: the voter's most recent vote, the adversary's vote if the voter was coerced, and the dummy votes (thus the dummy votes hide whether the voter submitted an additional ballot after being coerced). With R_j known, the submitted $\mathsf{Enc}(\sigma)$ can be compared directly to the exact roster entry with a plaintext equality test, avoiding the quadratic search.

Remarks on SKHS. This approach utilizes the fact that the voter knows which where in the roster she is, and if she can communicate this without harming her anonymity,

then the quadratic comparision between the submitted votes and the roster entries can be eliminated. Our system, Selection, in Chapter 11 works on the same principle but uses proofs instead of encrypted values. A drawback of the Spycher *et al.* approach is that it requires a *secret* number of dummy votes to be added for each voter, but offers no universally-verifiable method for ensuring the trustees added any or an amount from the right distribution. This is an interesting open problem.

AFT. The first (unbroken) linear-time coercion-resistant scheme was proposed by Araújo, Foulle and Traoré [AFT07, AFT10]. AFT uses a different approach than JCJ in registering voters. Instead of maintaining a public roster, voters are issued signed credentials where the signature attests to the voter's eligibility. The trustees distributively generate to threshold secret keys x and y and publish corresponding public keys $PK_x = g^x$ and $PK_y = g^y$. Voters are issued a credential of the form $\langle r, a, a^y, a^{x+xry} \rangle$ (*cf.* [CL04]) over an untappable channel along with a designated verifier proof of correct form. Note that without knowledge of x and y, one cannot determine if it has the correct form or not. Also note that for a random l, $\langle r, a^l, (a^y)^l, (a^{x+xry})^l \rangle$ has a valid form as well. Therefore the voter can generate many unlinkable credentials from the issued credential by using a different l.

Let v be the voter's selection and π_1 be the NIZKP that $\mathsf{Enc}(v)$ encrypts a valid candidate. The voter submits $\langle \mathsf{Enc}(v), \pi_1, a, \mathsf{Enc}(a^r), \mathsf{Enc}(a^{yr}), \mathsf{Enc}(a^{x+xry}), h^r, \pi_2 \rangle$ where h is an election-specific generator and π_2 is a set of NIZKPs: proofs of knowledge of plaintexts for each encryption and a proof of knowledge of r in h^r and $\mathsf{Enc}(a^r)$. If the voter is coerced, the voter supplies $\langle \hat{r}, a, (a^y), (a^{x+xry}) \rangle$ for a fake value $\hat{r}$ (and simulates the DV-proof). The vote can then cast her real ballot using r and $\langle a^l, (a^y)^l, (a^{x+xry})^l \rangle$. Denote the submitted tuple as $\langle t, \pi_1, u, v, w, z, h^r, \pi_2 \rangle$

The trustees eliminate submissions with improper proofs π_1 or π_2, use the h^r value to eliminate duplicate votes cast with the same r (a coerced vote will have a different r), and then shuffle the remaining submissions with a verifiable mix network. To check if a credential is actually signed with x and y, they compute v^y, prove y matches the public key, and use a plaintext equality test to determine if this equals w. They then compute $(uw)^x$, prove x matches the public key, and use a plaintext equality test to determine if this equals z. If both these tests pass, $\mathsf{Enc}(v)$ is decrypted and the vote is included in the tally. Araújo *et al.* have since proposed a modified version of AFT with different credentials and a security proof [ARR+10].

Remarks on AFT. The one drawback of AFT is that once a credential is issued, it is not possible to (directly) revoke the credential if the voter becomes ineligible to vote. This could arise if a voter moves, is incarcerated, or if the voter alleges to have forgotten

their credential and wants to be issued a new one (but may instead vote twice). In a roster-based system like JCJ and Civitas, the voter's encrypted credential can be crossed off or replaced.

Limited coercion-resistance. Wen and Buckland propose a system that offers ballot secrecy and a limited form of coercion-resistance [WB09a]. As in JCJ, voters obtain a private credential σ over an untappable channel and a designated verifier proof that it matches $\mathsf{Enc}(\sigma)$ on the roster. Similar to Acquisti's scheme, a voter computes a vote for v as $\mathsf{Enc}(v - \sigma)$ and submits it over an authenticated channel. The voter's roster entry can then be used to homomorphically compute $\mathsf{Enc}(v) = \mathsf{Enc}(v - \sigma) \cdot \mathsf{Enc}(\sigma)$ and votes can be tallied according to some underlying scheme (it is left abstract by the authors; but *e.g.,* mix, decrypt, and tally). If coerced to cast a certain vote, the voter claims her credential is the random value σ', which results in the random value $\mathsf{Enc}(v' - \sigma + \sigma')$ being cast for the adversary's vote v'. If coerced after casting a vote, she can claim to have submitted any value v' with fake credential σ' where $v' = (v - \sigma) + \sigma'$. Unlike in systems like JCJ and AFT where a coerced voter can still cast the vote of her choice, here she can only prevent the adversary from submitting a vote of his choice.[2] Under the definition, it is coercion-resistant with respect to spoiling a ballot.

Multiple-cast. Another form of limited coercion-resistance are the multiple-cast systems used in Estonia and elsewhere [VG06]. In these systems, voters can update their submitted vote at any time, overwriting the previous value. Kutyłowski and Zagórski propose a system where voters can cancel a previous ballot in a privacy-preserving way and submit a new ballot [KZ07]. Spycher, Haenni and Dubuis consider hybrid systems where voters can cast an in-person ballot after a period of internet voting, overwriting an votes they submitted online [SHD10]. We say multiple cast systems have limited coercion-resistance because a voter will not always be able to accomplish their intent, *e.g.,* if they are coerced at the end of the election or attending the polling place is observable to an adversary.

9.5 Other Issues in Internet Voting

Board flooding attacks. Koenig, Haenni, and Fischli observe that in coercion-resistant systems like JCJ and AFT, an adversary can flood the board with fake votes that are well-structured enough to not be excluded until after the first round of mixing (or later) [KHF11].

[2] Unless the voter knows the candidate the coercer will ask her to vote for. If she knows this before revealing her credential to the coercer, she can construct a fake credential such that $\sigma' = (v - \sigma) + v'$ where v' is the adversary's choice and v is hers. If the adversary casts a vote for v' with this credential, it will count for v.

In JCJ, this is particularly problematic since the talliers must do a computation that is quadratic in the number of submitted votes before the votes can be eliminated. Koenig *et al.* solve this essentially by issuing a finite number of tokens that are required to submit a vote. Voters receive a random number of tokens so they can still deceive a coercer. The details are more complex (the token and credential are combined into a single unit) but that is the intuition.

Anonymity revocation. Smart and Ritter note that in the United Kingdom, the ability to link a ballot to the voter who cast it is a legal requirement [SR09]. This is to trace votes that are suspected to have been cast fraudulently (*e.g.,* by someone other than the voter). They propose a protocol that allows individual votes to have their anonymity revoked. Although it is not referenced here, the idea of revocable anonymity has been studied in the general mix network setting [ABHO06] which may lead to alternative approaches for voting systems based on mixing.

Chapter 10

Panic Passwords: Authenticating under Duress

This chapter is adapted from published work supervised by Urs Hengartner [CH08].

10.1 Introductory Remarks

As important services and sensitive data congregate online, attackers have an increasing incentive to obtain the passwords that protect these services and data. Panic passwords are a mechanism to allow a user to use a special type of password to signal to the server that her password is being entered as the product of a coercive action. While panic passwords are currently used in home security systems to trip a silent alarm, they could also find application online. For example, a shortcoming of Internet-based voting is the move from a private voting booth to an open environment where voters could be coerced or bribed to demonstrably vote a certain way. A panic password could allow a voter to cast a coerced ballot as if it were her real vote, while in reality the ballot is spoiled by the server.

This chapter is motivated by a deficit of academic literature exploring the topic of panic passwords, the underlying threat models panic passwords address, and possible schemes to mitigate the threat while being usable. In particular, it is perturbing that the best-known example of a panic password scheme, one where the authenticator knows two passwords: a 'regular' one and a 'panic' one, is very easily defeated by coercing the victim to authenticate twice using different passwords each time.

Contributions. The contributions of this chapter can be summarized as:

- a thorough threat model for categorizing scenarios of coercion and undue influence,
- the application of an iteration and forced-randomization principle to panic passwords,
- the precise criteria for using the known scheme `2P`,
- the introduction of three new panic password schemes: `2P-lock`, `5-dictionary`, and `5-click`.

To make our contributions concrete, we will consider several representative scenarios where panic passwords could be employed. Our ultimate interest, however, is in Internet-based voting and in Chapter 11, we will use the results of this chapter to build an incoercible internet voting system based on passwords.

10.2 Related Work

Panic passwords are alternatively referred to as *distress passwords* or *duress codes* in the academic, commercial, and military literature. Due to their applicability in military and intelligence scenarios, it is not possible to determine the exact extent to which panic password schemes have been studied, as any such work would be classified. However a survey of the non-classified literature reveals very little.

Panic password schemes show up in patented technology, usually as one feature of a larger system. The exact method is often not included. A selection of patents include: ATMs which display a list of words from which the user may choose a predefined one for normal authentication and any other word to signal duress and to limit the amount of cash available for the transaction [BHYA04]; a home security system that incorporates a basic panic password scheme [MEL86]; the use of panic passwords to authenticate over a network where panic mode relays the user to a different database than the normal one [MP07]; the use of a panic password to keep data protected on a mobile device [Rus05]. Chaum mentions duress codes in passing for preventing coerced transactions using a credential [Cha92]. In a policy recommendation, the *Foundation for Information Policy Research* suggest that files hidden on a hard-disk (using a steganographic file system [ANS98]) could be erased upon entering a panic password [FIP06].

10.3 Security Model

Each of the scenarios considered herein involves three participants: Alice, Bob, and Oscar. Alice is typically a human and wishes to communicate, typically for the purpose of authentication prior to gaining access to a resource, to Bob, who may be human, a server, or a software application. Oscar is an opponent in the system and he attempts to coerce

Alice's communications with Bob in order to benefit his nefarious purposes. We will also consider less coercive scenarios where Alice wishes to sell her access rights to Oscar. We assume that Bob is a trusted entity and, in particular, is not in collusion with Oscar.

10.3.1 Passwords and Responses

Consider the password space P, where each $p \in P$ is in one of two sets, V for valid or I for invalid. As in any password scheme, a regular password p_0 is selected from P and is included in V. Entering an invalid password results in an error (observable to Alice and Oscar) and has no further consequence. To extend the notion of passwords to panic passwords, we define the following concepts:

- **Panic Password**: a covert communication from Alice to Bob over an observed channel indicating an abnormal state. We define a panic password as $p^* \in P$ and include it in V.
- **Panic Password Scheme**: a scheme specifies a password space P and a partition of P into I and V. It provides two services: registration and authentication. Registration allows a user to select any $p_0 \in P$ as their regular password and the scheme maintains p_0. Registration may also allow a user to select any number of $p^* \in P$ as panic passwords. Alternatively, the residue set V/p_0 could contain the set of panic passwords. Authentication accepts a submitted password p_i at time t_i and produces a decision (which is split into a reaction and response defined below) based on whether $p_i \in V$ and whether $p_i = p_0$. A scheme may also maintain state, such as past values of $\langle p_i, t_i \rangle$, and use these values in its decision.
- **Unobserved Reaction**: in reaction to receiving a p^* from Alice, Bob may engage in an unobserved reaction $q^* \in Q$, where Q is the set of all such reactions and $q_0 \in Q$ is the regular (non-panic) reaction. This reaction is unobservable to Alice or Oscar. A scheme will include a single regular reaction and any number of panic reactions (*i.e.,* none, one, or multiple reactions).
- **Observable Response**: in reaction to receiving p^* from Alice, Bob may respond to Alice with a panic response $r^* \in R$, where R is the set of all responses and $r_0 \in R$ is the regular response. This response is observable to Alice and Oscar in principle, however it is not necessarily distinguishable to Oscar which response is which. A scheme will include a single regular response and any number of panic responses.

Consider again Internet-based voting. To authenticate to the voting service (Bob), assume Alice has a set of two passwords: a normal password and a single panic password, $\{p_0, p^*\}$. Entering p_0 will cause Alice's vote to be cast correctly, while entering p^* could trigger an unobserved reaction from Bob, q^*: for example, Bob could mark the ballot as

spoiled or could alert the authorities. Entering p^* could also trigger an observable response r^*: Bob may falsely inform Alice that her vote has already been cast and she may not modify it, or he may report that the server is down and unable to accept votes at this time. In other areas where panic passwords are used, an unobserved reaction could be a silent alarm, while an observable response may be modified access control permissions or an upper limit on funds available for a financial transaction.

A typical transaction will take the form,

$$A \rightarrow B \quad : \quad p \in P \tag{10.1}$$

$$B \quad : \quad q \in Q \tag{10.2}$$

$$A \leftarrow B \quad : \quad r \in R \tag{10.3}$$

10.3.2 Security Assumptions

Our threat model assumes the following principles, each of which affords Oscar a certain ability:

- **Kerckhoffs' principle**: The details of the authentication system that is in place is public information and known to all participants, including Oscar. Alice and Bob retain a shared secret (*e.g.,* a password or set of passwords) that is both private and the foundation upon which the security of the system rests.
- **Observational principle**: Alice communicates with Bob over a semi-private channel that can be observed by Oscar prior to the password being secured (*i.e.,* encrypted or hashed). Namely, Oscar can observe the password(s) used by Alice and could even enter in Alice's password himself, after coercing her into revealing it.
- **Iteration principle**: Unless explicitly prevented by the underlying system, Oscar is not bound to a single instance of coercion against Alice. He may force Alice to authenticate multiple times. Combined with the observational principle (Assumption 2), Oscar can force Alice to use a different password each time.
- **Forced-randomization principle**: Oscar can choose to eliminate any strategy Alice may employ through the order in which she reveals the passwords she knows. For example, Oscar may force Alice into writing down a set of passwords so that he can randomly choose the order in which to iterate through them. The assumption is that this option is available to Oscar, not that he will necessarily employ it.

10.3.3 Threat Model Parameters

The underlying threat model is dependent on the scenario where a panic password system is employed. We identify four parameters along which the threat model could vary. They

Parameter 1	Parameter 2
persistent	$\{p^*, q^*, r_0\}$
non-persistent	$\{p^*, q_0, r^*\}$
	$\{p^*, q^*, r^*\}$

Parameter 3	Parameter 4
$G_{\mathrm{O}}[\forall q \in T : \neg q^*]$	screening
$G_{\mathrm{O}}[\exists q \in T : q_0]$	signaling
$G_{\mathrm{O}}[\forall r \in T : \neg r^*]$	
$G_{\mathrm{O}}[\exists r \in T : r_0]$	

Table 10.1: Summary of parameters to the threat model.

are summarized in Table 10.1.

Parameter 1: Oscar's Persistence. We can define the period of time under which Alice is expected to be coerced by Oscar as beginning at time t_0. In some scenarios, Oscar may be *non-persistent* and limited to time interval $[t_0, t_1]$, as in the example of Oscar coercing Alice into withdrawing funds from an ATM. In other, typically less-threatening, scenarios, Oscar may be *persistent* in his coercion for an arbitrarily long period of time, as in the example of a employer influencing the vote of her employee. A panic password scheme that disables access to an account could deter a non-persistent attacker if the account were disabled for a period of time greater than t_1 but this measure would not effectively deter a persistent attacker.

Parameter 2: Bob's Reaction and Response. We can summarize Equations 10.1 to 10.3 by defining a transaction as a tuple of parameters, such as $\{p_0, q_0, r_0\}$ to mean Alice submits non-panic communication and Bob reacts with the regular reaction and response. If Alice presents a panic password, Bob has three options. He could perform an unobserved reaction but not change his observable response, $\{p^*, q^*, r_0\}$; he could modify his response but not commit any unobserved reaction, $\{p^*, q_0, r^*\}$; or he could do both, $\{p^*, q^*, r^*\}$.

Parameter 3: Oscar's Goal. Oscar's goal, denoted G_{O}, in coercing Alice may take different forms. His proximate goal is for Alice to enter p_0 but since he reserves the option of iteration, we must distinguish between a few scenarios. These distinctions are dependent on his ultimate goal with respect to Parameter 2.

If Bob performs a panic reaction to a panic password, Oscar could have one of two goals: to prevent this unobserved reaction from ever occurring in the set of transactions

T, $G_{\mathrm{O}}[\forall q \in T : \neg q^*]$; or to achieve at least one regular reaction, $G_{\mathrm{O}}[\exists q \in T : q_0]$. If Oscar is limited to a single transaction, $|T| = 1$, these goals are equivalent. However under the Iteration Principle, Oscar may force Alice to authenticate more than once and these goals are no longer equivalent when $|T| > 1$. If q^* were a silent alarm, Oscar would have the first goal of preventing any occurrence of q^*. However if q^* was the reaction of spoiling a ballot in an Internet-based election, Oscar may have the second goal: he could force Alice to re-authenticate many times with different passwords and vote each time, hoping that eventually she enters p_0 and this transaction will be counted as a valid vote.

If Bob responds differently to a panic password than to a regular password in parameter 2, Oscar also has one of two goals: to prevent any observable panic response $G_{\mathrm{O}}[\forall r \in T : \neg r^*]$ or to achieve at least one regular response $G_{\mathrm{O}}[\exists r \in T : r_0]$. As an example of the first, consider the case where Alice has data hidden in an encrypted partition on her harddrive. The regular response could reencrypt this data and then provide access to it, while a panic response could overwrite the most sensitive data with random data and provide access to only the less sensitive data. Oscar's goal would be to prevent a panic response, as he cannot recover from such a state. Alternatively, a panic response could be less permissive access control permissions than is regular, and Oscar's goal will be to apply iteration until he gets full access.

Parameter 4: Screening vs. Signalling. A proper panic password scheme should cause Alice's use of a panic password to be indistinguishable by Oscar from use of her valid password. If Oscar coerces Alice into using her regular password, he succeeds by being able to distinguish (or convince Alice he can distinguish) the use of a panic password. Alice succeeds by not having to enter her regular password. Any mechanism that allows Oscar to separate panic and regular passwords is called a *screen*. Alternatively, Alice may want to prove to Oscar she is not using a panic password. Here Alice's goal is to demonstrate to Oscar that she is entering a regular password, and any mechanism she can use to accomplish this is called a *signal*. In Internet-based voting, voter-coercion is based on screening, whereas vote-selling is based on signalling. Although there is no duress in vote-selling schemes, the availability of panic passwords to prevent voter-coercion offers the additional feature that Oscar cannot determine if he is actually buying a legitimate vote from Alice, who could cheat him with her panic password. Thus, signal prevention is often a free additional property of panic password schemes.

Summary. As summarized in Table 10.1, our threat model has four parameters. Parameters 1 and 4 have two options, while parameter 2 has three, and parameter 3 has four. This set of parameters allow for 48 unique scenarios where panic passwords could be used,

however some combinations of parameters are arbitrary or not meaningful.[1] The others reduce to general classes of attack. In the next section, we consider four classes of scenarios which constitute each meaningful class of attack possible under these parameters.

10.4 Illustrative Threats and their Prevention

We now consider several illustrative scenarios where panic passwords can be applied. These scenarios were chosen to represent a class of specific threat models to which a particular panic password scheme applies.

10.4.1 Unrecoverable Reactions

Consider a scenario with panic communication $\{p^*, q^*, r_0\}$ such that $G_{\mathrm{O}}[\forall q \in T : \neg q^*]$. Our approach to this problem is invariant with respect to the persistence of Oscar or the prevention of signals as opposed to screens. The approach also applies in an equivalent manner to the scenarios: $\{p^*, q_0, r^*\}$ such that $G_{\mathrm{O}}[\forall r \in T : \neg r^*]$ and $\{p^*, q^*, r^*\}$ such that $G_{\mathrm{O}}[\forall q \in T : \neg q^*]$ or $G_{\mathrm{O}}[\forall q \in T : \neg q^*]$.

Working example. Consider the scenario where Alice is an employee at an office building with a security alarm that must be deactivated with a numerical password (entrance code) upon entry. Alice wants to communicate a forced entry which alerts the alarm company or law enforcement (q^*) but still grants Oscar access to the building (r_0). Oscar's goal is to prevent the silent alarm as he cannot escape on time to avoid detention.[2]

Solution. Security alarms on the market already handle this through a system we will call `2P` for two passwords.

Scheme 1. *`2P`. Alice knows two passwords $\{p_0, p^*\}$. For normal authentication purposes, Alice uses p_0. To communicate panic, she uses p^*.*

In the working example, Alice could use a four digit PIN, $d_0d_1d_2d_3$, as p_0 and invert the order of the last two digits to construct p^*: $d_0d_1d_3d_2$. Under our first security assumption, Kerckhoffs' principle, Oscar is assumed to know the structure of the system while

[1] For example, if Bob's response to a panic state is $\{p^*, q_0, r^*\}$ and Oscar's goal is $G_{\mathrm{O}}[B : q_0]$ or $G_{\mathrm{O}}[B : \neg q^*]$, he will always achieve his goal.

[2] To demonstrate why Oscar's goal is $G_{\mathrm{O}}[\forall q \in T : \neg q^*]$ and not $G_{\mathrm{O}}[\exists q \in T : q_0]$, consider a situation where Oscar forces Alice to authenticate twice, once yielding q^* and once q_0. This outcome is unacceptable to Oscar if he has the former goal but is acceptable for the latter goal. If q^* is a silent alarm, this outcome is unacceptable and thus Oscar must not have the latter goal in this scenario.

not knowing p_0 or p^*. From the observational principle, Oscar can observe Alice enter a password, p_1, but cannot determine if it is p_0 or p^*. If access is granted, Oscar can learn that it was one of the two passwords. Since Oscar's goal is to prevent the silent alarm, q^*, forcing Alice to enter a second password after entering the first (assumption 3) will only assure he does not achieve his objective. Under these three security assumptions, the 2P scheme succeeds at mitigating the threat.

The fourth assumption, however, presents a problem. Since Oscar knows the scheme being used, he can ask Alice to reveal the two passwords she knows. He cannot distinguish them but instead of him allowing Alice to enter the password of her choosing, which presumably would be p^*, he can randomly choose one. This allows him to achieve his goal half of the time on average. However this probability could be reduced by having a larger set of panic passwords, as will be examined below, and the significant probability of being caught that 2P provides is likely a large enough deterrent for scenarios in this class.

10.4.2 Non-persistent Attacks

Consider a scenario where Oscar has a non-persistent influence and his goal is either $G_{\mathrm{O}}[\exists q \in T : q_0]$ or $G_{\mathrm{O}}[\exists r \in T : r_0]$. This scenario differs from the previous one since Oscar is also not attempting to prevent a q^* or r^*, only to gain a clean q_0 or r_0 as the situation dictates (thus, the outcome in footnote 2 would be acceptable).

Working example. Consider the scenario where Alice is forced to withdraw money from an ATM by Oscar. In addition to wanting to signal a silent alarm (which we assume Oscar is not trying to prevent, as he believes he can escape on time), the ATM issues marked bills in place of real money. Oscar has a method for distinguishing marked bills but cannot do it on the spot. Therefore, his goal is only to escape with unmarked money even if he is not initially sure which bills are which.

Solution. The 2P system is inadequate in this scenario as Oscar can conduct an attack using the iteration principle (assumption 3). He will first force Alice to type in her PIN and withdraw money, and then knowing the 2P system is in place (assumption 1), he will force her to authenticate a second time while observing that she enters a different PIN (assumption 2). Alice will have no choice but to enter p_0 on one of the two occasions and Oscar will achieve his goal.

To address this threat, we can exploit the fact that Oscar is non-persistent and must end his coercion at some point, t_1, to escape. Specifically if Alice's account is temporarily locked until some time $t_2 > t_1$, Oscar will not achieve his goal. We must, however, exercise caution in how locking of an account is triggered. A naïve approach may be to lock the

account after receiving p^*, however Oscar could use this information to screen p_0 from p^*. He could demand that Alice enters a password that does not produce a locked account by threatening retaliation against her if her password does, thereby accomplishing his goal of getting unmarked money. For this reason, the event that locks the account must be invariant to the type of the passwords being entered.

Scheme 2. `2P-lock`. *Alice knows two passwords $\{p_0, p^*\}$. For normal authentication purposes, Alice uses p_0. To communicate panic, she uses p^* which causes r^* or q^* as the scenario dictates (but does not not lock the account). If Alice authenticates multiple times within a window of time, t_1, using the same password each time, her account will not be locked. However upon using two different passwords within t_1, the account will be made inaccessible for a period of t_2 such that $t_2 > t_1$.*

The `2P-lock` scheme is designed specifically to thwart an iteration attack. Once Alice has entered one password, forcing her to enter the other password is futile as it will only lock the account. Furthermore, the locking will occur regardless of whether Alice initially entered p_0 or p^*—making it impossible for Oscar to determine if he achieved his goal by the behaviour of the scheme. Oscar can still conduct a forced-randomization attack, giving him probability $1/2$ of receiving unmarked money, however the scheme could include more than one panic password which would lower this probability. Also, Oscar could coerce Alice when he knows she has had recent (less than t_1 ago) communications with Bob. Since Oscar knows Alice has recently entered p_0, he knows that any password that locks the account is a panic password, giving him the same screening and coercion abilities that he had under `2P`. This may be useful if Oscar is a remote adversary, however if Oscar is physically present and could observe when Alice communicates with Bob, he could more simply hijack her account after she has authenticated and before she logs out.

10.4.3 Persistent Attacks

Consider a scenario with panic communication $\{p^*, q^*, r_0\}$ and a persistent adversary with goal $G_{\mathrm{O}}[\exists q \in T : q_0]$. In addition, preventing both signals and screens is important.

Working example. Consider the previously defined improper influence problem in internet voting. Alice could be coerced by Oscar into voting a certain way. Alternatively, Alice may want to sell her vote to Oscar by casting her ballot in his presence. Bob's observable response will be the report of a successful casting of Alice's vote, but he will take the unobserved reaction of disregarding any votes cast under a panic password.

Solution. In this case, we assume Oscar is persistent, meaning he can observe Alice for extended periods of time. In the working example, he could be her employer, an intrusive partner, or a union representative. For this reason, locking the account until t_2 is not guaranteed to be effective. It is also not prudent as it would easily allow an adversary to prevent Alice from voting by coercing her into repeatedly locking herself out of her account—an effective denial of service attack. Therefore `2P-lock` is not suitable. The nature of the system does not preclude the iteration attack either, rendering `2P` ineffective as well.

The problem with a persistent attacker is that he could eventually exhaust Alice's memory of panic passwords, forcing her to enter p_0 eventually. Therefore a suitable scheme for this model would equip Alice with an arbitrarily large number of panic passwords, so that she could produce a virtually endless list of them.

Scheme 3. *`P-Complement`. Alice knows one password, p_0, which she uses for normal authentication purposes. To communicate panic, she uses any other password: $\{\forall p \neq p_0 \in P : p^*\}$. There are no invalid passwords in this scheme.*

This system allows Alice to enter any password other than her real password to communicate a panic state. The drawback of `P-Complement` is that if Alice mistakenly enters the wrong password, by definition of the problem, she cannot distinguish the panic state from the normal one. From a usability perspective this is undesirable. A solution could be to have Alice enter and then reenter her password, as is common during password registration. Alternatively, we could attempt to reduce the impact of typos with a different design.

If we generalize `P-Complement`, Alice knows two things: one real password p_0 and a rule, '*anything except p_0 is a panic password.*' The fundamental problem is that the rule is too inclusive. We can generalize to four desirable properties,

1. an arbitrarily large number of panic passwords,
2. an arbitrarily large set of invalid passwords,
3. a small distance between panic and invalid passwords,
4. a rule that is cognitively easy to apply.

The reason for the first property is twofold: we wish to prevent an iteration attack without locking the account, and as a password scheme, it must have all the security properties of any password system. As such, it must not be susceptible to exhaustive search attacks. The second property is to promote the probability that a mistyped p_0 is an invalid password instead of a panic password. This property is necessary but not sufficient—we also need to assure that panic and invalid password spaces are well mixed. For example,

a good metric for passwords is the Damerau-Levenshtein distance [Bar07], which is the minimum number of a specific set of operations to convert one string into another, where the operations are insertion, deletion, or substitution of a single character and transposition of two characters. The metric is designed for application to spell-checking. Finally, from a usability perspective, users need to apply the rule confidently, especially since they are operating under duress.

As a general approach, we start with a password space P and we apply the separating function $f()$ to each element in it such that any $p \in P$ is mapped to V or I (*i.e.,* valid or invalid). $|V|$ and $|I|$ should both be sufficiently large. p_0 is selected from V and a panic password is $\{p_v^* \in V | p_v^* \neq p_0\}$. All elements of I are invalid.

One implementation of this general scheme is used by some US governmental agencies [Car08]. A password is combined with a PIN to create p_0. Panic passwords are any combination such that the password portion is correct and the PIN is incorrect. Compositions of a wrong password are considered invalid. This scheme is problematic if a mistake is made on the PIN portion. Also, under coercion, Alice is forced to reveal the password part of p_0 in order to use a p^* which reduces the security of p_0 against an exhaustive search significantly. Since the threat model these passwords are used under is unknown by the authors, we are not suggesting this scheme is used improperly—only that it has drawbacks for scenarios of this type.

Scheme 4. *`5-Dictionary`. Let P be the set of all ordered lists of five strings and let V be the set of all ordered lists of five strings in a standard dictionary. Alice composes a password by selecting a set of five dictionary words. Any other set of five dictionary words is a panic password. Any other set of either dictionary or non-dictionary strings is considered invalid.*

This systems meets the five criteria listed above. The standard Unix dictionary is just over 25,000 words, which provides up to 73 bits of security for a 5-word combination. The invalid password space is arbitrarily large, as any word can be any other combination of alphanumerical characters. Although empirical data of common typos, along with numerical analysis would need to confirm the average Damerau-Levenshtein distance between typos and words, our intuition is that there is a considerable probability that many typos would be caught—certainly higher than in `P-Complement`. It remains open to investigate whether entering the same password twice with a scheme like `P-Compliment` has usability (or is better received by users) than entering it a single time with a scheme like `5-Dicitionary`. A similar approach to `5-Dicitionary` could be taken with a click-based graphical password scheme.

Scheme 5. *`5-Click`. Alice is presented with a sequence of five images, and a rule for what regions of the image are in V based on the content of the picture. She knows one password*

$\{p_0\}$ *which is a sequence of one click per image which she uses for normal authentication purposes. To communicate panic, she clicks on another valid region for at least one of her clicks.*

For example, in the PassPoints scheme [WWB+05], users authenticate by clicking on certain pixels (within a tolerance) in an image. A separating function could be used to predefine certain regions of an image for V and give them semantic meaning to help users remember them. For example, V could be faces on the cover image of *Sgt. Pepper* or cars in an image of a parking lot. Using a sequence of different images will increase entropy, and is similar to the Cued Click-Points (CCP) scheme [COB07], where users authenticate by clicking one point in each of a sequence of five images. However in CCP, the i^{th} image displayed to the user is dependent on where the user clicked in the (i-1)^{th} image. This last property has significant usability improvements, as it provides feedback to the user in case of an error, but in this model it could also be used by an adversary to screen—the user would know the sequence of images for p_0 and but not p^* (unless she explicitly committed a panic sequence to memory). Instead, using the same sequence of images would be a suitable compromise between increased entropy and usability.

10.4.4 Screening Observable Responses

For a final scenario, consider a scenario with $\{p^*, q_0, r^*\}$ in a screening model where persistent Oscar's goal is $G_{\text{O}}[\exists r \in T : r_0]$. In particular, our focus is on an inherent problem between r^* and screening. Our approach to this scenario applies equivalently to $\{p^*, q^*, r^*\}$ and $G_{\text{O}}[\exists q \in T : q_0]$.

Working example. Consider the scenario where Alice has network access to sensitive information, such as credit card account numbers, which are password protected. Under coercion, she wants the server to return false information that looks real or be routed through to a "honeypot" server [Pro04].

Solution. Consider using `2P`. Given that $r^* \neq r_0$, Oscar can apply the iteration principle to r instead of p. He can request that Alice authenticates to the server using a different password such that it produces a different r. Once he observes two different r values, he knows one must be r_0. The solution to this problem parallels the dictionary solution—the set of r must be arbitrarily large. For this reason, it is not always possible to solve this problem. The password-side of scheme can be handled by `5-Dictionary` but the response-side would have to contain randomization of data.

It may appear possible to compromise by having a finite but secret number of panic responses, such that Alice could claim that all possible responses have been iterated through when at least the real one still remains unseen by Oscar. There are two problems with this approach. First, it is a slight breach of assumption 1 depending on how you define what is part of the system and what is the shared secret. Second, if the panic state draws randomly from a finite set of possibilities, these states will eventually repeat while the legitimate data will not repeat.[3] For this reason, Oscar can screen out sets of data as illegitimate once they appear more than once and use this property to force Alice to authenticate with p_0.

We leave a generalized solution to scenarios of this nature for future work. Presenting randomized responses that appear legitimate depends on the what "appear legitimate" and are thus situational. It appears that solutions would have to be tailored for the specific circumstances.

10.5 Discussion

Issuing new panic passwords. In the case of panic passwords, the ability to be issued a password or reset a password on the basis of supplemental authentication information provides a backdoor that an adversary can successfully exploit. Instead of forcing Alice to authenticate, Oscar would simply force her to reveal the information necessary to reset the password or be issued a new one. Without completely solving the problem, passwords could be issued but not reset, effectively limiting Oscar's window of opportunity—this may be feasible in less critical scenarios. For more serious scenarios, there appears to be no alternative to having the user create and reset passwords in a controlled, coercion-free environment. This could be by visiting a physical bank to set an ATM PIN or online banking account, or a one-time in-person registration to enable online voting, which is likely an improvement over voting in person in every election on a specific day. In some countries, government services are provided under federated access and so the registration could be used in scenarios beyond voting.

Reconsidering Kerchhoffs' principle. In most scenarios, multiple individuals use the same system and so the assumption that the attacker knows the system is reasonable. However, there are scenarios where the adversary may not. For example, disk encryption utilities like TrueCrypt can create hidden partitions within an already encrypted partition of data. The hidden partition looks random—as does the unused space in the outer partition. If an adversary wishes to coerce Alice into revealing the data on her computer, we cannot assume Oscar knows that a hidden partition exists or can distinguish it from

[3]We assume a consistent username is used by Alice.

unused space. This appears to be a safe application of panic passwords outside of our threat model.

Reconsidering trustworthy Bob. When introducing the three participants in the threat model—Alice, Bob, and Oscar—we assumed Bob, the entity being authenticated to by Alice, was trustworthy. In certain scenarios, this is not an adequate assumption. For example, we considered the example of Internet-based voting where Bob's hidden reaction was to dispose of votes cast under a panic password and count only the votes cast under a normal password. However within the context of E2E verifiability, the system should provide proof that the real ballots are included in the tally and the panic ballots are discarded. Additionally, such a proof shoudl not be useful to Oscar in screening the ballots he forced a voter to cast. We present a solution to this problem in Chapter 11.

10.6 Concluding Remarks

Despite a large volume of research on the use of passwords, examined from many diverse angles—security, usability, agreement schemes, storage, dual factor, biometrics, to name a few—academic study of panic passwords is virtually non-existent. While they appear in some commercial and military applications, our hope is that this chapter provides a basis for further attention from the research community.

Future work. We believe that better schemes are possible and more involved threat models could be devised. In particular, attention from a usability perspective would be extremely valuable, including user studies and case studies.

Chapter 11

Selections: Coercion-Resistant Internet Voting

This chapter is adapted from published work supervised by Urs Hengartner [CH11].

11.1 Introductory Remarks

From a security perspective, the use of electronic voting machines in elections around the world continues to be concerning. In principle, many security issues can be allayed with cryptography. While cryptographic voting has not seen wide deployment, refined systems like Prêt à Voter [CRS05, RBH+09] and Scantegrity II [CCC+08] are representative of what is theoretically possible, and have even seen some use in governmental elections [CCC+10]. Today, a share of the skepticism over electronic elections is being apportioned to internet voting.[1] Many nation-states are considering, piloting or using internet voting in elections. In addition to the challenges of verifiability and ballot secrecy present in any election system, internet voting adds two additional constraints:

- Untrusted platforms: voters should be able to reliably cast secret ballots, even when their devices may leak information or do not function correctly.
- Unsupervised voting: coercers or vote buyers should not be able to exert undue influence over voters despite the open environment of internet voting.

As with electronic voting, cryptography can assist in addressing these issues. The study of cryptographic internet voting is not as mature. Most of the literature concentrates on

[1] One noted cryptographer, Ronald Rivest, infamously opined that "best practices for internet voting are like best practices for drunk driving" [Kan10].

only one of the two problems (see related work in Section 11.2). In this chapter, we are concerned with the unsupervised voting problem. Informally, a system that solves it is said to be coercion-resistant.

Contributions. Coercion-resistant, end-to-end verifiable internet election systems have been proposed [Acq04, AFT07, CCM08, JCJ05, Smi05, WAB07]. However, these systems all require the voter to remember cryptographic information after registration. Since the information is too long to memorize, authentication can be considered to be based on "something you have." Voters must prepare for the possibility of coercion by creating fake values, proofs, or transcripts. Our system works with passwords, "something you know," and it allows a voter to supply a panic password during ballot casting that can be created mentally in real time by the voter. In summary, our system provides:

- Password-based authentication and cognitive coercion-resistance,
- In-person registration that can be performed bare-handed,
- Tallying that is linear in the number of voters, and
- Efficient revocation of voters from the roster during and between elections.

We compare Selections to three systems: JCJ [JCJ05], Civitas [CCM08], and AFT [AFT07]. Of these properties, only Selections meets each while AFT achieves the third and both JCJ and Civitas achieve the fourth.

11.2 Comparison to Related Work

Our system addresses the problem of coercion and vote selling when voters are not required to vote in a private booth. As we showed in Chapter 9, only a small number of the most recent papers in cryptographic voting address this threat, and only a smaller subset of these are unbroken.

Coercion-resistance was first formalized by Juels *et al.*, who also provide a coercion-resistant system, often referred to as JCJ [JJ02, JCJ05]. JCJ was independently implemented as Civitas [CCM08]. The main drawback of both is that tallying is quadratic in the number of voters. Aquisti [Acq04] refined JCJ to use Paillier encryption and support write-in candidates, while both Smith [Smi05] and Weber *et al.* [WAB07] made the first attempts at reducing the complexity of tallying to linear. Unfortunately, all three are considered broken [AFT07, CCM08, ARR$^+$10]. Araujo *et al.* provide a linear-time system we refer to as AFT [AFT07]. More recently (concurrent with Selections), Spycher *et al.* have

proposed a different approach to making JCJ linear [SKHS11] (from our list of contributions, SKHS achieves the third and fourth). See Section 9.4 for further details on each of these proposals.

Both JCJ/Civitas and AFT provide registered voters with anonymous credentials. A voter submits a credential along with her vote and a procedure for computing a fake credential is provided (but cannot be done without a computer). In JCJ/Civitas, the credentials of registered voters are posted and these are anonymously and blindly compared to the credential accompanying each submitted vote. In AFT, the credentials of registered voters are essentially signed and the presence of a valid signature on a credential submitted during casting is anonymously and blindly checked. Due to the difficulty of revoking a signed value, voters cannot be revoked in AFT without a change of cryptographic keys.

11.3 Preliminaries

11.3.1 Selections: High-Level Overview

Selections is a protocol designed to allow voters to cast ballots over the internet during a window of time prior to traditional in-person voting. Voters can opt out of Selections at any time prior to election day and cast a ballot in person.

To be eligible for Selections, voters first complete a one-time, in-person registration protocol in a private booth without needing their own computational device. After this registration, the voter can vote in future elections over a tappable channel (see Section 11.3.3). The registration involves the voter choosing a password to be used for vote casting. However this password is non-traditional—it is a password from a panic password system (see Section 11.3.5). A semantically-secure homomorphic encryption of this password is posted on a public roster. The roster has an entry for each registered voter containing this ciphertext. The voter must be convinced that her entry is a correct encryption without being able to prove what it encrypts to anyone.

During vote submission, the voter asserts what her password is: it may be her actual password or a panic password. The voter creates a binding commitment to this asserted password. The voter then rerandomizes her entry off the roster. The voter selects a random subset of encrypted passwords off the roster and proves in zero-knowledge that her rerandomized entry is a rerandomization of one the ciphertexts in this set without revealing which one. The commitment to her asserted password, re-encrypted roster entry, proof (and some additional proofs that things are well-formed), and an encryption of her vote are submitted over an anonymous channel to a public bulletin board.

When the voting period expires, a distributed group of trustees will eliminate submissions with invalid proofs, eliminate duplicate votes based on the password commitment,

and then use a verifiable mix network to shuffle the order of the remaining submissions. After shuffling, voters can no longer determine where their submission is in the new permuted list. For each submission, the trustees will determine if the asserted password matches the roster entry without revealing either. If it does not, the entry is eliminated. The output of Selections is a list of encrypted votes from registered voters without duplicates. The entire protocol can be verified for soundness.

11.3.2 Coercion-resistance

Informally, Juels *et al.* define coercion-resistance as providing receipt-freeness, while preventing three attacks: randomization, abstention, and simulation [JCJ05]. An election system is said to be receipt-free if the voter cannot produce a transcript that constitutes a sound argument for how they voted [BT94]. Adversaries should not be able to force a registered voter to cast a random vote or to abstain from voting. Finally, the system should protect against voters surrendering their credentials and allowing a coercer or vote buyer to cast their vote for them. The dominant approach to preventing such a simulation is providing voters with the ability to create fake credentials. If an adversary cannot distinguish a real credential from a fake one, he will only be willing to pay what a fake credential is worth, which is nothing.

11.3.3 Untappable Channels

The main challenge for coercion-resistant internet voting is dealing with the elimination of the private voting booth, modelled as an untappable channel. One approach is to use multiple secure channels and assume that while any individual channel can be tapped, no adversary can tap all channels simultaneously. The second is to use an untappable channel just once, and bootstrap the output of this interaction into an arbitrary number of future interactions over secure (or anonymous) channels. We use the latter approach.

11.3.4 Registration Authority

In most coercion-resistant internet election systems, voters interact with a distributed registration authority [Acq04, AFT07, JCJ05]. To achieve coercion-resistance, it is assumed that at least one registrar is not corrupted by the adversary. Voters may be corrupted to retain a transcript, however the transcript has deniability by using a designated verifier proof [JSI96].

While distributing trust is usually an effective approach for achieving correctness and secrecy in a protocol, it is more complex with coercion-resistance. The voter must be

aware of which entity she trusts, so she can fake a proof that will not be compared to the original. If the voter discloses her private key to an adversary, it only requires a single malicious registrar to collude with the adversary and undetectably issue the voter an incorrect credential share (while retaining the correct value for potential adversarial use).

These concerns leave it unclear if the benefits of a distributed registration authority are worthwhile. While Selections is amenable to a distributed registration authority (voters would submit encryptions of shares of their password, which are homomorphically combined to create an encryption of the password), we describe the protocol using a single registrar that is assumed to not collude with a coercer (but may still misbehave in any other regard).

11.3.5 Panic Passwords

Recall from Chapter 10 the 5-Dictionary panic password system. Admissible passwords consist of five words from an agreed upon dictionary: the user chooses one combination as her password and any other combination is a panic password. A typo is likely to mutate the intended word into a string not found in the dictionary. We use this system in Selections.

11.4 The Selections Protocol

Selections involves four participants: a set of voters, a set of election trustees, an election authority, and a registrant. The system has six main protocols: registration set-up, voter preparation, registration, election set-up, casting, and pre-tallying. Let $\langle \mathsf{DKG}, \mathsf{Enc}, \mathsf{DDec} \rangle$ be a threshold encryption scheme. Distributed key generation $\mathsf{DKG}(n, t)$ generates public key, e, and a private key share, d_i, for each of n trustees. Encryption, $\mathsf{Enc}_e(m, r)$, is semantically secure and homomorphic with respect to one operation. Distributed decryption, $\mathsf{DDec}_{d_i}(c)$, on ciphertext c can be performed with $t+1$ trustees submitting shares d_i.[2] We use threshold Elgamal [Ped91], which is CPA-secure [TY98].

11.4.1 Registration Setup

The registration set-up protocol involves a set of n trustees: $\mathcal{T}_1, \ldots, \mathcal{T}_n$ and the election authority. Primes p and q are chosen such that the DL-problem and DDH-problem are hard

[2]Proactive security can maintain the secrecy of the shares over time, both the number of shares and the threshold can be adjusted without a dealer, and more a complex access structure than t-out-of-n can be created.

in the multiplicative subgroup $\mathbb{G}_q$ of $\mathbb{Z}_p^*$. Each T_j participates in $\mathsf{DKG}(n, m)$. Commitments are sent to the election authority, who posts them to an append-only broadcast channel called the Bulletin Board. At the end of the protocol, each $\mathcal{T}_j$ has private key share d_j and public key e is posted. The protocol is standard and will not be described here [Ped91].

11.4.2 Voter Preparation

The voter preparation procedure is performed by each voter $\mathcal{V}_i$ on a trusted computational client. Let $\langle P, I \rangle$ be the domain of a panic password system. P represents the set of admissible passwords and $I = \neg P$ is the set of inadmissible passwords. $\mathcal{V}_i$ chooses a password $\hat{\rho}$. The client runs $\mathsf{PassSubmit}(\hat{\rho})$, which tests if $\hat{\rho} \in P$. If $\hat{\rho} \in I$, $\mathsf{PassSubmit}(\hat{\rho})$ returns an error. The set of panic passwords are the remaining passwords in P: $\{\forall \hat{\rho}^* \in P | \hat{\rho}^* \neq \hat{\rho}\}$. $\mathsf{PassSubmit}(\hat{\rho^*})$ will behave identically upon submission of a panic password (otherwise an adversary could distinguish the case where he is given a panic password).

Once $\mathsf{PassSubmit}(\hat{\rho})$ accepts $\hat{\rho}$, the client encodes $\hat{\rho}$ as a bitstring and appends a non-secret salt to prevent accidental collisions with other users. This string is supplied as input to a password-based key derivation function (PBKDF) for strengthening and encoding into $\mathbb{Z}_q^*$. For brevity, we denote this entire password processing procedure as ϕ: $\rho \leftarrow \phi(\hat{\rho}) = \mathsf{PBKDF}(\mathsf{PassSubmit}(\hat{\rho}) \| \mathtt{salt})$.

Perhaps through a user-guided tutorial familiarizing the voter with the system, the voter will generate α admissible passwords: $\hat{\rho}_1, \ldots, \hat{\rho}_\alpha$. The value of α will determine the soundness of the registration protocol. An example value for α is 10. The password the voter wishes to register is in a random location in the list. Each is encrypted by the voter under the trustees' public key e. The voter prints out the list of ciphertexts on to a piece of paper, *e.g.,* with the ciphertexts encoded into barcodes. The registration protocol in Algorithm 8 includes the voter preparation protocol.

11.4.3 Registration

The registration protocol is completed by each voter $\mathcal{V}_i$. It is a two-party cut-and-choose protocol between a voter $\mathcal{V}_i$ and the registrar $\mathcal{R}$. The protocol is described in Algorithm 8. It is an adaptation of the Benaloh's voter initiated auditing [Ben06], with a predetermined number of challenges. The voter enters the protocol with a list of α encrypted passwords $\{c_1, \ldots, c_\alpha\}$ and the protocol completes with a re-encryption of one of the ρ's being posted to an append-only broadcast channel, called the Roster. The protocol itself is conducted over an untappable channel which is instantiated as an in-person protocol.

The voter presents identification and is authorized to register. The voter is given a blank transcript card and enters a private booth that has a computer in it capable of

Algorithm 8: Registration Protocol

Participants: Voter $\mathcal{V}_i$ and registrant $\mathcal{R}$
Public Input: Encryption parameters p, q, g, public key e, and soundness parameter $\alpha > 1$
Private Input (V_i): Ciphertexts $\{c_1, \ldots, c_\alpha\}$ as described below

Prior to the protocol, each voter should:

for *k from 1 to* α **do**
1. Choose a password $\hat{\rho}_k$.
2. Process password: $\rho_k \leftarrow \phi(\hat{\rho}_k)$.
3. Encrypt g^{ρ_k} with random r_k: $c_k \leftarrow \mathsf{Enc}_e(g^{\rho_k}, r_k)$.
4. Complete a NIZKP of knowledge of plaintext g^{ρ_k}: $\pi_k \leftarrow \text{NIZKP}_{pok}\{(\rho_k, r_k) : c_k = \mathsf{Enc}_e(g^{\rho_k}, r_k)\}$.
5. Record $\langle c_k, \pi_k \rangle$.

Registrar should:

6. Receive $\{\langle c_1, \pi_i \rangle, \ldots, \langle c_\alpha, \pi_\alpha \rangle\}$.
7. **for** *k from 1 to* α **do**
8. Check π_k.
9. Rerandomize c_k with random r'_k: $c'_k \leftarrow \mathsf{ReRand}(c_k, r'_k)$.
10. Print $\langle c'_k, (c_k, r'_k) \rangle$.

Each voter should:

11. Receive for each k: $\langle c'_k, (c_k, r'_k) \rangle$.
12. Optionally, rewind to line 7.
13. Choose $s \leftarrow [1, \alpha]$.
14. Erase (c_s, r'_s).
15. Send s to R.

Registrar should:

16. Receive s.
17. Publish $\langle \texttt{VoterID}, c'_s \rangle$ on the Roster.

Each voter should:

18. After leaving, check that $c'_k \leftarrow \mathsf{ReRand}(c_k, r'_k)$ for all $k \neq s$.
19. Check that received c'_s matches $\langle \texttt{VoterID}, c'_s \rangle$ on the Roster.

Remarks: This protocol is completed *bare-handed* [RT07] with pre-computations and erasures. The proof of knowledge of an Elgamal plaintext is standard. The cut-and-choose mechanism is a variant of Benaloh's voter initiated auditing [Ben06]. The option to rewind is included to prevent coercion contracts (see Section 5.4.5 and D.1).

printing and scanning barcodes. A transcript card has α rows and two columns. The second column for each row has a scratch-off surface. The voter is provided the option of downloading and printing a document from the internet—with the intention that the voter could print her voter preparation sheet in the event that an adversary ensured she entered the registration process without her sheet. The computer has a barcode scanner, which the voter uses to submit her α ciphertexts.

The computer will rerandomize each ciphertext and print the value in the first column

of the transcript card. Beside this value on the scratch-off surface, it will print the original ciphertext and the randomization used. The voter chooses one password to register: for that password, the voter will erase the original ciphertext and randomization by scratching off the appropriate cell.[3] It is assumed the voter cannot memorize, copy, or record a copy of the randomization. The voter shreds her preparation sheet and retains the transcript card. The remaining $\alpha - 1$ re-encryptions can be shown to anyone and checked for correctness at home.

11.4.4 Election Set-up

The Roster is a universal registration. To prepare for an election, entries from the Roster are copied to smaller lists, called ElectionRosters. An ElectionRoster is specific to a particular election, precinct or district. The trustees will also modify the encrypted message in each entry from g^ρ to g_0^ρ, where g_0 is a unique publicly-known generator for that election. This prevents information leakage across elections.

Recall that Roster entries are encrypted with ρ in the exponent: $\{c_1, c_2\} = \{g^r, g^\rho y^r\}$. For each ElectionRoster, each trustee chooses $b_i \leftarrow_r \mathbb{G}_q$. Then each trustee will in turn blind each ciphertext (with the same b_i) on the ElectionRoster as follows: output g^{b_i}, $c_1^{b_i}$ and $c_2^{b_i}$, and prove knowledge of b_i such that $g, c_1, c_2, g^{b_i}, c_1^{b_i}, c_2^{b_i}$ form a threewise DH-tuple with a NIZKP (*cf.* [CP92]). The next trustee will repeat the process using the previous trustee's output as input. All outputs are posted to an appendix on the ElectionRoster. Let $b_0 = \prod b_i$ and $g_0 = g^{b_0}$. The blinding sequence re-randomizes each ciphertext from r to $r' = r \cdot b_0$ and changes the encrypted message from g^ρ to g_0^ρ. The public and private key shares are the same. The public value g_0 will be used during the casting protocol.

11.4.5 Casting

The casting protocol involves a voter $\mathcal{V}_i$ and the election authority. The protocol is described in Algorithm 9. The communication occurs over an anonymous channel. The anonymity is to be built into the voter's client using an anonymous remailer or onion routing technology.

$\mathcal{V}_i$ submits a commitment to her asserted (*i.e.,* real or panic) password, $g_0^{\rho^*}$, and a rerandomization of her entry on the ElectionRoster, c'. If ρ^* matches the ρ encrypted in c', the pre-tallying protocol will ensure the ballot is included in the final result. Otherwise if it does not match, it will be discarded in a way that is unlinkable to the original submission.

[3] Under each scratch-off could be a pre-committed code in the form of a barcode, which the voter could scan to prove to the system that she scratched off the correct cell. We leave the details for such an augmented transcript card for future work.

Algorithm 9: Casting Protocol

Participants: Voter $\mathcal{V}_i$ and election authority

Public Input: Encryption parameters g, p, q, election parameter g_0, public key e, ElectionRoster, and anonymity parameter β

Private Input (V_i): Password (either real or panic) $\hat{\rho}^*$

Each voter should:

1. Find c for her `VoterID` from ElectionRoster.
2. Rerandomize c with random r: $c' \leftarrow \mathsf{ReRand}(c, r)$.
3. Randomly select β-1 other c_k from the ElectionRoster.
4. Form set $\mathcal{C} = \{c, c_1, \ldots, c_{\beta-1}\}$ in order of appearance on ElectionRoster.
5. Generate a NIZKP that r rerandomizes 1-out-of-β of $\mathcal{C}$.
   $\pi_1 \leftarrow \text{NIZKP}_{pok}\{(r) : c' = (\mathsf{ReRand}(c, r) \vee \mathsf{ReRand}(c_1, r) \vee \ldots)\}$.
6. Encode asserted password into $\mathbb{Z}_q^*$: $\rho^* \leftarrow \phi(\hat{\rho}^*)$.
7. Commit to ρ^*: $g_0^{\rho^*}$.
8. Complete an NIZKP of knowledge of ρ^*:
   $\pi_2 \leftarrow \text{NIZKP}_{pok}\{(\rho^*) : g_0, g_0^{\rho^*}\}$.
9. Complete a ballot and retain ballot information $\mathbf{B}$.
10. Send $\left\langle g_0^{\rho^*}, c', \mathbf{B}, \pi_1, \pi_2 \right\rangle$ to A.

Authority should:

11. Publish $\left\langle g_0^{\rho^*}, c', \mathbf{B}, \pi_1, \pi_2 \right\rangle$ on AllVotes.

Remarks: Rerandomization proofs are formed with a knowledge of a DDH-tuple proof due to Chaum and Pedersen [CP92]. 1-out-of-m proofs are due to a heuristic by Cramer, Damgard and Schoenmakers [CDS94]. Proof of knowledge of a discrete log is due to Schnorr [Sch91]. Parameter β represents the voter's anonymity set.

$\mathcal{V}_i$ must prove that c' is from the ElectionRoster. Simply including her entry without rerandomizing it reveals that she submitted a vote. To prevent abstention attacks, she instead rerandomizes it, draws an additional $\beta-1$ entries randomly from the ElectionRoster, and proves in zero-knowledge that c' is a rerandomization of one of these β entries (her entry plus the additional ones). β acts as an anonymity set. Most voters will use a small value of β, however privacy-conscious voters can also (at extra computational cost) cast a stealth vote where β includes all the entries on the ElectionRoster.

Selections is designed to be versatile with different options for capturing and tallying the votes themselves. Thus we leave the information the voter submits with regard to their vote abstractly as $\mathbf{B}$ while only requiring that $\mathbf{B}$ is submittable to a mix-network. For example, $\mathbf{B}$ could be an encryption of the preferred candidate(s) or a tuple of cryptographic counters for each option, accompanied by proofs of validity as appropriate. Note that our coercion-resistance guarantee extends only to the delivery of valid, eligible, and unique $\mathbf{B}$ values, and care should be taken to ensure that tallying these values does not break coercion-resistance.

Each ZKP uses the Fiat-Shamir heuristic to make it non-interactive, and each uses the values $\left\langle g_0^{\rho^*}, c', \mathbf{B} \right\rangle$ in creating the challenge. This prevents an adversary from replaying any of the proofs individually. The submission is posted to an append-only broadcast channel called AllVotes.

If the voter is under coercion, she makes up a panic password and follows the rest of the protocol as specified. She can later cast a stealth vote with her real password. If a voter wants to overwrite a previous vote submitted under password ρ^*, the inclusion of the same $g_0^{\rho^*}$ will indicate in cleartext that it is an overwrite. Therefore, she should use the same β entries from the ElectionRoster as her anonymity set. Also note that the inclusion of the same $g_0^{\rho^*}$ across multiple elections would also be linkable if the value g_0 was not changed in each election.

11.4.6 Pre-tallying

The pre-tallying protocol involves an authorized subset of the N election trustees. The protocol is described in Algorithm 10. The protocol takes AllVotes and produces a shorter list of only the most recently cast votes for voters that supply the correct, registered password. Checking the validity of each vote is linear in β. For these voters, the list includes just the ballot information, $\mathbf{B}$, in an order that is unlinkable to the order of submission. How this list is further processed to produce a tally is dependent on the election system our system interfaces with (which is why this is called a pre-tally). In a simple case, $\mathbf{B}$ is an encryption of the voter's selections (with a proof of knowledge) and the final step is jointly decrypting each $\mathbf{B}$ from the list.

11.4.7 Voter Revocation

Between elections, Selections offers a way of choosing which registered voters are eligible or not to vote in a particular election. In Selections, it is also possible to revoke a voter at any point before the pre-tallying protocol. This could arise because the voter forgot their password (and is issued a new one) or registered to vote online but decides to vote in person. For every submitted vote that includes the revoked voter among its β registered voters in its anonymity set (which will include any potentially valid vote by the revoked voter herself), the submitted password is checked against the revoked voter's entry on the ElectionRoster using a plaintext-equality test. Revocation of this type is the same in Civitas and is not possible in AFT. Coercion-resistance does not necessarily extend to all types of revocation.

Algorithm 10: Pre-Tallying Protocol

Participants: Authorized set of trustees $\mathcal{T}_1, \ldots, \mathcal{T}_t$ and election authority
Public Input: AllVotes
Private Input (T_i): Share of private key, d_i
Authority should:
1 For each entry, check π_1 and π_2.
2 Remove all tuples with invalid proofs to form list ProvedVotes
3 Find all entries in ProvedVotes with duplicate values for g_0^ρ.
4 Remove all but the most recent to form list UniqueVotes.

Each participating trustee should:
5 Participate in verifiable mix network for shuffling UniqueVotes.
Note: the initial $g_0^{\rho^*}$ is treated as $c_\rho = \mathsf{Enc}_e(g_0^{\rho^*}, 0)$.
6 Output is AnonUniqueVotes.

Each participating trustee should:
7 **for** *each entry in AnonUniqueVotes* **do**
8 Read entry $\langle c_\rho, c', \mathbf{B} \rangle$.
9 Participate in a plaintext-equality test of c_ρ and c':
$\{\mathtt{T}, \mathtt{F}\} \leftarrow \mathrm{PET}_{d_i}(c_\rho, c')$.

Authority should:
10 Remove all tuples with PET outcome of `False` to form list ValidVotes.

Each participating trustee should:
11 **for** *each entry in ValidVotes* **do**
12 Participate in threshold decryption of $\mathbf{B}$.

Remarks: Various protocols exist for verifiable mix networks. An efficient technique with statistical soundness is randomized partial checking [JJR02]. The plaintext equality test (PET) is due to Juels and Jakobsson [JJ00] (see Section 2.3.5). The output of this protocol is the ballot information for unique and registered voters in an order that is unlinkable to the order of submission.

11.5 Security Analysis (Abstract)

11.5.1 Soundness of Registration

In Appendix D, we show that the Registration protocol is a cut-and-choose argument for $\{(c, r) : c' = \mathsf{ReRand}_e(c, r)\}$. It takes soundness parameter α (*e.g.,* $\alpha = 10$). It is complete and has statistical soundness of $1 - \alpha^{-1}$ for a single run. After k runs, soundness increases to $1 - \alpha^{-k}$. Designing a bare-handed argument with stronger soundness (*e.g.,* $1 - 2^{-\alpha}$ for a single run) is open. With erasures, the protocol has deniability for c and computational secrecy for r.

The protocol does not protect against covert channels. This has been addressed in the literature with verifiable random functions [GGR09] or pre-committed randomness [FB09].

The protocol protects against coercion contracts [CHL09] with rewinds. Rewinds can be eliminated if the voter commits to their choice of password at the beginning of the protocol.

11.5.2 Coercion-Resistance

In Appendix D, we show several results concerning the coercion-resistance (cr) of Selections. Juels *et al.* define an experiment $\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}$ for non-adaptive adversary $\mathcal{A}$ in election system ES, as well as an ideal $\mathbf{Exp}^{\text{cr-ideal}}_{ES,\mathcal{A}}$. The critical component in $\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}$ is a coin flip $b \leftarrow_r \{0,1\}$ defining a corrupted voter's behaviour. If $b = 0$, the voter provides (in Selections) a panic password to the adversary and casts a vote with her real password. If $b = 1$, the voter complies with the adversary and provides her real password. In both cases, the adversary can use the supplied password to submit a vote. We define the advantage of $\mathcal{A}$, where an output of 1 is the adversary correctly stating b, as,

$$\mathbf{adv}^{\text{cr}}_{ES,\mathcal{A}} = |\mathbf{Pr}[\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}(\cdot) = 1] - \mathbf{Pr}[\mathbf{Exp}^{\text{cr-ideal}}_{ES,\mathcal{A}}(\cdot) = 1]|.$$

Case 1: $\beta = R$**.** We show that when β is the full roster R, $\mathbf{adv}^{\text{cr}}_{ES,\mathcal{A}}$ for Selections is negligible. Setting $\beta = R$ does impact performance. Vote casting is linear in the size of the ElectionRoster and Pre-Tallying is quadratic. However the only quadratic component is checking the 1-out-of-β rerandomization proof, where the proof length is linear in the size of the roster. These proofs can be pre-checked, while voters submit votes.

Case 2: $\beta = \text{const}$**.** We show that when β is constant (*e.g.,* 5 or 100), $\mathbf{adv}^{\text{cr}}_{ES,\mathcal{A}} < \delta$, where δ is small but non-negligible. Recall there are V_2 votes with valid proofs and R entries on the ElectionRoster. Let $\mathbf{F}(k; p, n)$ be the cumulative distribution function of a Binomial distribution with n trials, success probability p, and k successes. We show that δ for this case is,

$$\delta \;=\; \frac{1}{2}(\mathbf{F}(\frac{\beta V_2}{R}; V_2, \frac{\beta}{R}) + 1 - \mathbf{F}(\frac{\beta V_2}{R} - 1; V_2 - 1, \frac{\beta}{R})).$$

Case 3: $\beta \geq \text{const}$**.** Finally we consider the case where β is required to be at least a constant value (*e.g.,* 5 or 100) but voters can submit stealth votes where $\beta = R$. We show that if a corrupted voter's coercion-resistant strategy is to submit their real vote as a stealth vote, $\mathbf{adv}^{\text{cr}}_{ES,\mathcal{A}}$ is negligible. We do make one small change to $\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}$: instead of the corrupted voter's real vote being appended to the cast ballots, it is inserted at a random place (*i.e.,* she votes her real ballot at some arbitrary time independent of when she is coerced).

		Civitas	AFT	Selections
Registration	Registrar	7	9	2α
	Voter	11	10	4α-1
Casting	Voter	10	24	$(2\beta + 9)$
Pre-Tally	Check Proofs	$4V_0$	$20V_0$	$(4\beta + 6)V_0$
	Remove Duplicates	$(1/2)(V_1^2 - V_1)(8T + 1)$	—	—
	Check Removal	$(1/2)(V_1^2 - V_1)(8T + 1)$	—	—
	Mix	$8V_2T + 4RT$	$20V_2T$	$12V_2T$
	Check Mix	$4V_2T + 2RT$	$10V_2T$	$6V_2T$
	Remove Unregistered	$(8A + 1)V_2R$	$(16T + 8)V_2$	$(8T + 1)V_2$
	Check Removal	$(8A + 1)V_2R$	$(16T + 10)V_2$	$(8T + 1)V_2$

Table 11.1: Comparison of the efficiency of the main protocols in Civitas, AFT, and Selections, measured with modular exponentiations.

11.6 Performance

We compare the performance of Selections to JCJ as implemented in Civitas [CCM08] and to AFT [AFT07]. We make a number of standardizing assumptions to facilitate a better comparison. We assume a single registrar, T trustees, R registered voters, and V_0 submitted votes. We do not use the "blocking" technique of Civitas, which could improve the performance of all three systems. Of the V_0 submitted votes, $V_1 \leq V_0$ have correct proofs, $V_2 \leq V_1$ are not duplicates, and $V_3 \leq V_2$ correspond to registered voters. Recall that for Selections, α are the number of submitted ciphertexts in registration and β is the size of the voter's anonymity set during casting.

We use Elgamal encryption in each system, with proofs of knowledge of plaintexts where appropriate. We assume each trustee participates in decryption (*i.e.,* distributed instead of threshold). We assume that ballot material is encrypted with only a proof of knowledge (no additional proofs of well-formedness). The pre-tallying protocol ends with a list of V_3 encrypted ballots. Finally, we assume mixing is done with a re-encryption mixnet and randomized partial checking [JJR02], where each authority produces two mixes and half of these re-encryptions are checked. The complete details of our comparison are in Appendix D.3.

Table 11.1 shows the efficiency in terms of modular exponentiations and Figure 11.1 shows a comparison of the pre-tallying protocols. With full forced-abstention, Selections is quadratic like Civitas but with a smaller constant. When β is a constant, Selections is linear in the number of submitted votes like AFT. The exact value of β dictates which is exactly faster. Recall our goal was not to improve the efficiency of AFT but rather to create a password-based system with similar performance to AFT. To this end, we are successful.

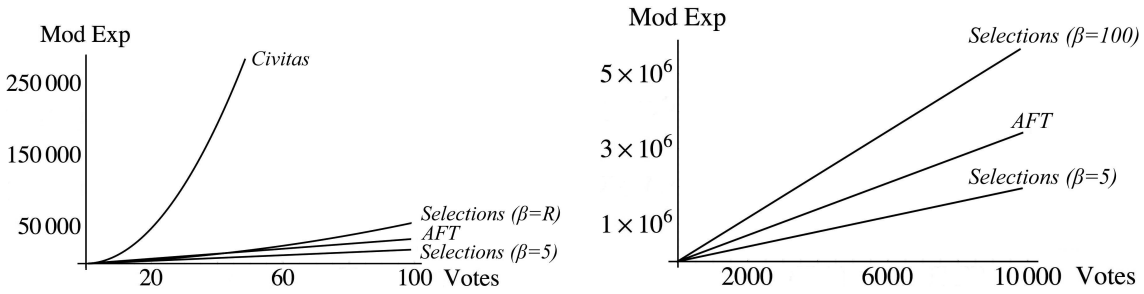


Figure 11.1: Pre-tallying efficiency in modular exponentiations with $T = 5$ and variable $R = V_0 = V_1 = V_2$.

11.7 Concluding Remarks

Selections has many benefits: users can evade coercion without computations, registration does not require a computer, tallying the votes is linear in the number of voters, and voters can have their registration efficiently revoked.

Future Work. The next major step for Selections, and other coercion-resistant internet voting schemes, is providing protection against untrusted platforms. This could perhaps be done by merging Selections with the existing work on code voting we reviewed in Chapter 9.

Chapter 12

Concluding Remarks

The goal of this dissertation is to convince the reader that the work being done on end-to-end verifiable election systems has the potential to enhance democracy. The exciting properties of these systems, in combination with the attempts to increase their deployability and policy compatibility, could help prevent fraud, detect faults, and disincentivise coercion in elections.

A word of caution, however, is due. While important work, this is by no means a comprehensive solution to democratic problems. There are many ways to manipulate an election outside of the vote capture process. Some of these are *legal*: voter suppression through targeted identification and registration laws, gerrymandering, vote dilution, introducing spoiler candidates to bias the result against the Condorcet candidate, or voter suppression by distributing inadequate equipment creating wait lines in targeted districts. Other manipulations are *illegal*: spreading misinformation about the election day, harassing voters, and impersonating voters. Creating confidence in the integrity of our democratic process is a problem in need of a comprehensive solution.

While cryptographic techniques may not be able to solve everything, we argue that they can help resolve two of the most important: the integrity of the tally and the secrecy of the ballot.

References

[AB96] R. J. Anderson and E. Biham. Two practical and provably secure block ciphers: BEARS and LION. In *FSE*, 1996. 114

[ABDV03] R. Aditya, C. Boyd, E. Dawson, and K. Viswanathan. Secure e-voting for preferential elections. In *EGOV*, 2003. 36

[ABHL03] L. v. Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *EUROCRYPT*, 2003. 155

[ABHO06] L. v. Ahn, A. Bortz, N. J. Hopper, and K. O'Neill. Selectively traceable anonymity. In *PET*, 2006. 161

[ACC+08] A. Aviv, P. Cerny, S. Clark, E. Cronin, G. Shah, M. Sherr, and M. Blaze. Security evaluation of ES&S voting machines and election management system. In *EVT*, pages 1–13, 2008. 2

[ACG06] R. Araujo, R. F. Custodio, and J. v. Graaf. A verifiable voting protocol based on Farnel. In *WOTE*, 2006. 44

[ACG10] R. Araujo, R. F. Custodio, and J. v. Graaf. A verifiable voting protocol based on Farnel. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*. Springer, 2010. 44

[Acq04] A. Acquisti. Receipt-free homomorphic elections and write-in ballots. Technical report, IACR Eprint Report 2004/105, 2004. 158, 177, 179

[Adi08] B. Adida. Helios: web-based open-audit voting. In *USENIX Security Symposium*, pages 335–348, 2008. 36, 40, 87, 93, 137, 155

[AFT07] R. Araujo, S. Foulle, and J. Traoré. A practical and secure coercion-resistant scheme for remote elections. In *Frontiers of Electronic Voting*, 2007. 158, 159, 177, 179, 188, 246

[AFT10] R. Araujo, S. Foulle, and J. Traoré. A practical and secure coercion-resistant scheme for internet voting. *Toward Trustworthy Elections*, LNCS 6000, 2010. 158, 159, 246

[AGH+09] A. W. Appel, M. Ginsburg, H. Hursti, B. W. Kernighan, C. D. Richards, G. Tan, and P. Venetis. The new jersey voting-machine lawsuit and the avc advantage dre voting machine. In *EVT/WOTE*, 2009. 2

[AJL04] A. Ambainis, M. Jakobsson, and H. Lipmaa. Cryptographic randomized response techniques. *PKC*, 2004. 146

[ALBD04] R. Aditya, B. Lee, C. Boyd, and E. Dawson. An efficient mixnet-based voting scheme providing receipt-freeness. In *TrustBus*, 2004. 34

[AMPQ09] B. Adida, O. d. Marneffe, O. Pereira, and J.-J. Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of Helios. In *EVT/WOTE*, 2009. 36, 93, 134, 137

[AN06] B. Adida and C. A. Neff. Ballot casting assurance. In *EVT*, 2006. 39

[AN09] B. Adida and C. A. Neff. Efficient receipt-free ballot casting resistant to covert channels. In *EVT/WOTE*, 2009. 12, 39

[ANS98] R. J. Anderson, R. M. Needham, and A. Shamir. The steganographic file system. In *IH*, 1998. 163

[Ans09] S. Ansolabehere. Guide to the 2008 cooperative congressional election survey. Technical report, M.I.T., 2009. 1

[AR06] B. Adida and R. L. Rivest. Scratch & Vote: self-contained paper-based cryptographic voting. In *ACM WPES*, pages 29–40, 2006. 42

[AR08] R. Araujo and P. Y. A. Ryan. Improving the Farnel voting scheme. In *EVOTE*, 2008. 44

[ARR+10] R. Araujo, N. B. Rajeb, R. Robbana, J. Traoré, and S. Yousfi. Towards practical and secure coercion-resistant electronic elections. In *CANS*, 2010. 158, 159, 177

[AW03] R. Aggarwal and G. Wu. Stock market manipulation—theory and evidence. *Journal of Business*, 79(4), 2003. 97

[Bab85] L. Babai. Trading group theory for randomness. In *ACM STOC*, 1985. 91

[Bar07] G. V. Bard. Spelling-error tolerant, order-independent pass-phrases via the Damerau-Levenshtein string-edit distance metric. In *AISW*, 2007. 172

[BBS86] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo random number generator. *SIAM J. Comput.*, 15(2):364–383, 1986. 124

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988. 14

[BCE+07] M. Blaze, A. Cordero, S. Engle, C. Karlof, N. Sastry, M. Sherr, T. Stegers, and K.-P. Yee. Source code review of the sequoia voting system. In *State of California's Top to Bottom Review*, 2007. 2

[BDJR97] M. Bellare, A. Desai, E. Jokiph, and P. Rogaway. A concrete security treatment of symmetric encryption: an analysis of the DES mode of operation. In *IEEE FOCS*, 1997. 134

[BEH+08] K. Butler, W. Enck, H. Hursti, S. McLaughlin, P. Traynor, and P. McDaniel. Systemic issues in the hart intercivic and premier voting systems: Reflections on project everest. In *EVT*, 2008. 2

[Ben86] J. Benaloh. Cryptographic capsules: A disjunctive primitive for interactive protocols. In *CRYPTO*, 1986. 29

[Ben87] J. Benaloh. *Verifiable secret-ballot elections*. PhD thesis, Yale University, 1987. 29

[Ben06] J. Benaloh. Simple verifiable elections. In *EVT*, 2006. 40, 87, 124, 181, 182, 227

[Ben07] J. Benaloh. Ballot casting assurance via voter-initiated poll station auditing. In *EVT*, 2007. 40, 87

[BF85] J. D. Benaloh (*né* Cohen) and M. J. Fisher. A robust and verifiable cryptographically secure election scheme. In *SFCS*, 1985. 11, 24, 25, 29

[BF05] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, 2005. 108

[BFP+01] O. Baudron, P.-A. Fouque, D. Pointcheval, G. Poupard, and J. Stern. Practical multi-candidate election system. In *ACM PODC*, 2001. 33

[BG02] D. Boneh and P. Golle. Almost entirely correct mixing with applications to voting. In *ACM CCS*, 2002. 26

[BGN05] D. Boneh, E.-J. Goh, and K. Nissam. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, 2005. 34

[BH05] B. Barak and S. Halevi. A model and architecture for pseudo-random generation and applications to `/dev/random`. In *ACM CCS*, 2005. 108

[BHYA04] L. C. Baird, M. E. Harmon, R. R. Young, and J. E. Armstrong. Apparatus and method for authenticating access to a network resource. United States Patent, 6732278, 2004. 163

[BMN+09] J. Benaloh, T. Moran, L. Naish, K. Ramchen, and V. Teague. Shuffle-sum: Coercion-resistant verifiable tallying for stv voting. *IEEE TIFS*, 4(4):684–698, 2009. 36

[BMR07] J. M. Bohli, J. Mueller-Quade, and S. Roehrich. Bingo voting: Secure and coercion-free voting using a trusted random number generator. In *VOTE-ID*, pages 111–124, 2007. 40, 73, 75, 79

[BO91] J. J. Bartholdi and J. B. Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8:341–354, 1991. 73

[Bor07] B. Borchert. Segment-based visual cryptography. Technical Report WSI-2007-04, WSI, 2007. 140

[Bos92] J. Bos. *Practical Privacy.* PhD thesis, Technische Universiteit Eindhoven, 1992. 28

[Bou00] F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, 2000. 20

[Boy89] C. Boyd. A new multiple key cipher and an improved voting scheme. In *EUROCRYPT*, 1989. 28

[BPS95] A. Baraani-Dastjerdi, J. Pieprzyk, and R. Safavi-Naini. A practical electronic voting protocol using threshold schemes. In *ACSAC*, 1995. 31

[Bra97] S. Brands. Rapid demonstration of linear relations connected by boolean operators. In *EUROCRYPT*, 1997. 18

[BS73] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3), 1973. 95

[BT94] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *ACM STOC*, 1994. 2, 10, 11, 23, 25, 29, 30, 179

[BT08] A. Broadbent and A. Tapp. Information-theoretically secure voting without an honest majority. In *WOTE*, 2008. 11, 35

[BY86] J. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters. In *ACM PODC*, 1986. 29

[Car08] R. T. Carback. Private Communications, 2008. 172

[CC97] L. F. Cranor and R. K. Cytron. Sensus: A security-conscious electronic polling system for the internet. In *Hawaii International Conference on System Sciences*, 1997. 36

[CCC+08] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman. Scantegrity II: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *EVT*, 2008. 45, 90, 93, 134, 142, 144, 176

[CCC+09] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, A. T. Sherman, and P. L. Vora. Scantegrity ii: end-to-end verifiability by voters of optical scan elections through confirmation codes. *IEEE Transactions on Information Forensics and Security*, 4(4):611–627, 2009. 45

[CCC+10] R. T. Carback, D. Chaum, J. Clark, J. Conway, A. Essex, P. S. Hernson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, A. T. Sherman, and P. L. Vora. Scantegrity II election at Takoma Park. In *USENIX Security Symposium*, 2010. 45, 141, 142, 144, 176

[CCEP07] R. T. Carback, J. Clark, A. Essex, and S. Popoveniuc. On the independent verification of a Punchscan election. In *Proc. VoComp*, 2007. 42

[CCM08] M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy*, pages 354–368, 2008. 156, 158, 177, 188

[CDG87] D. Chaum, I. Damgård, and J. v. Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO*, 1987. 14

[CDM00] R. Cramer, I. Damgård, and P. MacKenzie. Efficient zero-knowledge proofs of knowledge without intractability assumptions. In *PKC*, 2000. 17, 20

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994. 18, 184

[CEA07] J. Clark, A. Essex, and C. Adams. Secure and observable auditing of electronic voting systems using stock indices. In *IEEE Canadian Conference on Electrical and Computer Engineering*, 2007. 43, 87, 92, 93, 114, 135

[CEC+08] D. Chaum, A. Essex, R. Carback, J. Clark, S. Popoveniuc, A. T. Sherman, and P. Vora. Scantegrity: End-to-end voter verifiable optical-scan voting. *IEEE Security and Privacy*, 6(3):40–46, May/June 2008. 45

[CFH+07] J. A. Calandrino, A. J. Feldman, J. A. Halderman, D. Wagner, H. Yu, and W. P. Zeller. Source code review of the Diebold voting system. In *State of California's Top to Bottom Review*, 2007. 2

[CFSY96] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret-ballot elections with linear work. In *EUROCRYPT*, 1996. 11, 12, 29

[CG96] R. Canetti and R. Gennaro. Incoercible multiparty computation. In *IEEE FOCS*, 1996. 25

[CGS97] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT*, 1997. 12, 15, 25, 32, 36

[CH08] J. Clark and U. Hengartner. Panic passwords: Authenticating under duress. In *USENIX HotSec*, 2008. 162

[CH10] J. Clark and U. Hengartner. On the use of financial data as a random beacon. In *EVT/WOTE*, 2010. 89

[CH11] J. Clark and U. Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. In *FC*, 2011. 176

[Cha81] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981. 2, 25, 26, 27

[Cha82] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, 1982. 25

[Cha88] D. Chaum. Elections with unconditionally-secure ballots and disruption equivalent to breaking rsa. In *EUROCRYPT*, 1988. 11, 28

[Cha92] D. Chaum. Achieving electronic privacy. *Scientific American*, pages 96–101, 1992. 163

[Cha01] D. Chaum. Surevote: Technical overview. In *WOTE*, 2001. 154

[Cha02] D. Chaum. Secret-ballot receipts and transparent integrity: Better and less-costly electronic voting at polling places. Technical report, VReceipt, 2002. 38, 69

[Cha04] D. Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security and Privacy*, 2(1):38–47, 2004. 23, 38, 69, 140

[CHF08] J. A. Calandrino, J. A. Halderman, and E. W. Felten. In defense of pseudorandom sample selection. In *EVT*, 2008. 92

[CHK+06] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. How to win the clone wars: efficient periodic n-times anonymous authentication. In *ACM CCS*, 2006. 111, 215

[CHL09] J. Clark, U. Hengartner, and K. Larson. Not-so-hidden information: optimal contracts for undue influence in E2E voting systems. In *VOTE-ID*, 2009. 72, 187

[CJPY97] M. Chesney, M. Jeanblanc-Picque, and M. Yor. Brownian excursions and Parisian barrier options. *Advances in Applied Probability*, 29, 1997. 97

[CL04] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, 2004. 159

[CLW07] S. S. M. Chow, J. K. Liu, and D. S. Wong. Robust receipt-free election system with ballot secrecy and verifiability. In *NDSS*, 2007. 35

[COB07] S. Chiasson, P. C. v. Oorschot, and R. Biddle. Graphical password authentication using cued click points. In *ESORICS*, 2007. 173

[CP92] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO*, 1992. 18, 183, 184

[CRS05] D. Chaum, P. Y. A. Ryan, and S. Schneider. A practical voter-verifiable election scheme. In *ESORICS*, 2005. 40, 120, 148, 176

[CS97] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical report, ETH Zurich, 1997. 18

[CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provavly secure against adaptive chosen ciphertext attack. In *CRYPTO*, 1998. 17

[Cus01] R. F. Custodio. Farnel: um protocolo de votacao papel com verificabilidade parcial. In *Simposio Seguranca em Informatica*, 2001. 44

[CWD06] A. Cordero, D. Wagner, and D. Dill. The role of dice in election audits — extended abstract. In *WOTE*, 2006. 92, 135

[CWF+09] W. Clarkson, T. Weyrich, A. Finkelstein, N. Heninger, J. A. Halderman, and E. W. Felten. Fingerprinting blank paper using commodity scanners. In *IEEE Symposium on Security and Privacy*, 2009. 141

[DDWY93] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *Journal of the ACM*, 40(1):17–47, 1993. 153

[DF89] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO*, 1989. 16

[DGH+04] Y. Dodis, R. Gennaro, J. Hastad, H. Krawczyk, and T. Rabin. Randomness extraction and key derivation using the CBC, Cascade and HMAC modes. In *CRYPTO*, 2004. 107, 108, 111, 210, 212

[DGS03] I. Damgård, J. Groth, and G. Salomonsen. The theory and implementation of an electronic voting system. In *Secure Electronic Voting*, 2003. 34

[Dia08] L. Diamond. *The Spirit of Democracy: The Struggle to Build Free Societies Throughout the World*. Times Books, first edition, 2008. 1

[DJ01] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *PKC*, pages 119–136, 2001. 17, 34

[DJ02] I. Damgård and M. Jurik. Client/server tradeoffs for online elections. In *PKC*, pages 125–140, 2002. 13, 34, 155

[DKR06] S. Delaune, S. Kremer, and M. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *CSFW*, 2006. 11

[DLM82] R. DeMillo, N. Lynch, and M. J. Merritt. Cryptographic protocols. In *ACM STOC*, 1982. 35

[DNW96] B. Davenport, A. Newberger, and J. Woodward. Creating a secure digital voting protocol for campus elections. Technical report, Princeton, 1996. 36

[Dur64] R. Durstenfeld. Algorithm 235: Random permutation. *Communications of the ACM*, 7(7):420, 1964. 124

[DY05] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *PKC*, 2005. 109, 111, 210, 211, 215

[Eas04] D. Eastlake. Publicly verifiable nominations committee (nomcom) random selection. RFC 3797, IETF, 2004. 93

[Eas06] D. Eastlake. Publicly verifiable nomcom random selection. RFC 2777, IETF, 2006. 93

[ECA08] A. Essex, J. Clark, and C. Adams. Aperio: High integrity elections for developing countries. In *WOTE*, 2008. 44, 122, 148

[ECA10] A. Essex, J. Clark, and C. Adams. Aperio: High integrity elections for developing countries. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*. Springer, 2010. 44, 122, 148

[ECCP07a] A. Essex, J. Clark, R. T. Carback, and S. Popoveniuc. Punchscan in practice: an e2e election case study. In *WOTE*, 2007. 45, 90, 93, 134

[ECCP07b] A. Essex, J. Clark, R. T. Carback, and S. Popoveniuc. The Punchscan voting system. Technical report, Submission to the *University Voting Systems Competition (VoComp)*, 2007. 42

[ECHA09] A. Essex, J. Clark, U. Hengartner, and C. Adams. How to print a secret. In *HotSec*, 2009. 139

[ECHA10] A. Essex, J. Clark, U. Hengartner, and C. Adams. Eperio: Mitigating technical complexity in cryptographic election verification. In *EVT/WOTE*, 2010. 13, 116, 135

[EGL85] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 26(6), 1985. 91

[Ele06] Election Data Services Inc. Voting equipment summary by type as of 11/07/2006, 2006. 1

[Elg84] T. Elgamal. A publick key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, 1984. 15

[FB09] A. J. Feldman and J. Benaloh. On subliminal channels in encrypt-on-cast voting systems. In *EVT/WOTE*, 2009. 12, 40, 186, 230

[FCS06] K. Fisher, R. T. Carback, and A. T. Sherman. Punchscan system definition. In *WOTE*, 2006. 42

[Fed09] Federal constitutional court of Germany (Bundesverfassungsgericht). Judgement of 3 March 2009 – 2 BvC 3/07, 2 BvC 4/07 – Verfahren über die Wahlprüfungsbeschwerden, 2009. 1

[FIP06] FIPR. Consultation on the draft code of practice for the investigation of protected electronic information – part III of the regulation of investigatory powers act 2000. FIPR Response to the Home Office, 2006. 163

[FMM+02] J. Furukawa, H. Miyauchi, K. Mori, S. Obana, and K. Sako. An implementation of a universally verifiable electronic voting scheme based on shuffling. In *FC*, 2002. 36

[FMS10] J. Furukawa, K. Mori, and K. Sako. An implementation of a mix-net based network voting scheme and its use in a private organization. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*. Springer, 2010. 36

[FO98] E. Fujisaki and T. Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In *EUROCRYPT*, 1998. 35

[FOO92] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *ASIACRYPT*, pages 244–251, 1992. 31, 35, 36

[FPS00] P.-A. Fouque, G. Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries. In *FC*, 2000. 34

[FPS08] R. Farashahi, R. Pellikaan, and A. Sidorenko. Extractors for binary elliptic curves. *Designs, Codes, and Cryptography*, 49, 2008. 111

[FS86] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986. 19, 91, 135

[Gee01] D. Geer. Technical maturity, reliability, implicit taxes, and wealth creation. *login: The magazine of Usenix & Sage*, 26(8), 2001. 70

[Gen95] R. Gennaro. Achieving independence efficiently and securely. In *ACM PODC*, 1995. 12, 19

[GGR09] R. W. Gardner, S. Garera, and A. D. Rubin. Coercion resistant end-to-end voting. In *FC*, 2009. 12, 40, 186, 230, 231

[GHH+09] R. Gonggrijp, W.-J. Hengeveld, E. Hotting, S. Schmidt, and F. Weidemann. RIES—Rijnland internet election system: A cursory study of published source code. In *VOTE-ID*, 2009. 60

[GJKR99] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT*, 1999. 16

[GK03] S. Goldwasser and Y. Kalaj. On the (in)security of the Fiat-Shamir paradigm. In *IEEE FOCS*, 2003. 93

[Gro04a] J. Groth. Efficient maximal privacy in boardroom voting and anonymous broadcast. In *FC*, 2004. 35

[Gro04b] J. Groth. Review of RIES. Technical report, Cryptomathic, 2004. 60

[GS86] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In *ACM STOC*, 1986. 91

[GS04] J. Groth and G. Salomonsen. Strong privacy protection in electronic voting. Technical Report RS-04-13, BRICS, 2004. 34, 155

[GS08] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, 2008. 91

[Hal08] J. L. Hall. On improving the uniformity of randomness with alameda county's random selection process. Technical report, UC Berkeley, 2008. 92

[Her97] M. A. Herschberg. Secure electronic voting over the world wide web. Master's thesis, MIT, 1997. 36

[Hir01] M. Hirt. *Multi-Party Computation: Efficient Protocols, General Adversaries and Voting*. PhD thesis, ETH Zurich, 2001. 32, 33

[Hir10] M. Hirt. Receipt-free K-out-of-L voting based on ElGamal encryption. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*. Springer, 2010. 33

[HJS+08] E. Hubbers, B. Jacobs, B. Schoenmakers, H. van Tilborg, and B. de Weger. Description and analysis of the RIES internet voting system. Technical report, Eindhoven Institute for the Protection of Systems and Information (EiPSI), 2008. 60

[HL08] J. Heather and D. Lundin. The append-only web bulletin board. In *FAST*, 2008. 25

[HL10] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols*. Springer, 2010. 17

[HLL10] S. Heiberg, H. Lipmaa, and F. v. Laenen. On e-vote integrity in the case of malicious voter computers. In *ESORICS*, 2010. 155

[HMP95] P. Horster, M. Michels, and H. Petersen. Blind multisignature schemes and their relevance to electronic voting. In *ACSAC*, 1995. 31

[HS00] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT*, 2000. 25, 30, 32, 33, 34, 39, 153

[HS07] J. Helbach and J. Schwenk. Secure internet voting with code sheets. In *VOTE-ID*, 2007. 154

[HSS08] J. Helbach, J. Schwenk, and S. Schage. Code voting with linkable group signatures. In *EVOTE*, 2008. 154

[HSS09] K. Henry, D. R. Stinson, and J. Sui. The effectiveness of receipt-based attacks on threeballot. *IEEE TIFS*, 4(4):699–707, 2009. 44, 87

[HT88] M. A. Huang and S. Teng. Secure and verifiable schemes for election and general distributed computing problems. In *ACM PODC*, 1988. 35

[HW07] J. A. Halderman and B. Waters. Harvesting verifiable challenges from oblivious online sources. In *ACM CCS*, 2007. 93

[IRS+07] S. Inguva, E. Rescorla, H. Shacham, , and D. S. Wallach. Source code review of the Hart InterCivic voting system. In *State of California's Top to Bottom Review*, 2007. 2

[JCJ05] A. Juels, D. Catalano, and M. Jacobsson. Coercion-resistant electronic elections. In *ACM WPES*, 2005. 10, 11, 73, 156, 177, 179, 231

[JGM+07] D. Jefferson, E. Ginnold, K. Midstokke, K. Alexander, P. Stark, and A. Lehmkuhl. Evaluation of audit sampling models and options for strengthening California's manual count. Technical report, State of California's Post-Election Audit Standards Working Group, 2007. 92

[JJ00] M. Jakobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In *ASIACRYPT*, 2000. 21, 186

[JJ02] A. Juels and M. Jakobsson. Coercion-resistant electronic elections. Technical report, Citeseer, 2002. 177

[JJR02] M. Jacobsson, A. Juels, and R. L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security Symposium*, pages 339–353, 2002. 26, 59, 123, 186, 188

[JJSH00] A. Juels, M. Jakobsson, E. Shriver, and B. Hillyer. How to turn loaded dice into fair coins. *IEEE Transactions on Information Theory*, 46(3), 2000. 92

[JR07] R. Joaquim and C. Ribeiro. Codevoting: protection against automatic vote manipulation in an uncontrolled environment. In *VOTE-ID*, 2007. 154

[JRF09] R. Joaquim, C. Ribeiro, and P. Ferreira. Veryvote: A voter verifiable code voting system. In *VOTE-ID*, 2009. 154

[JRF10] R. Joaquim, C. Ribeiro, and P. Ferreira. Improving remote voting security with CodeVoting. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*. Springer, 2010. 154

[JSI96] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In *EUROCRYPT*, 1996. 20, 158, 179

[Kan10] C. Kane. Voting and verifiability: interview with Ron Rivest. *RSA Vantage Magazine*, 7(1), 2010. 176

[Kat03] J. Katz. Efficient and non-malleable proofs of plaintext knowledge and applications. In *EUROCRYPT*, 2003. 20

[Kat07] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, pages 115–128, 2007. 120

[KHF11] R. Koenig, R. Haenni, and S. Fischli. Preventing board flooding attacks in coercion-resistant electronic voting schemes. In *SEC*, 2011. 160

[Kim02] K. Kim. Killer application of pki to internet voting. In *International Workshop for Asia Public Key Infrastructures*, 2002. 36

[KK08] J. Keller and J. Kilian. A linked-list approach to cryptographically secure elections using instant runoff voting. In *ASIACRYPT*, 2008. 36

[KRMC10] J. Kelsey, A. Regenscheid, T. Moran, and D. Chaum. Attacking paper-based E2E voting systems. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*, pages 370–387. Springer, 2010. 73, 75, 79

[KS04] J. G. Koppella and J. A. Steena. The effects of ballot position on election outcomes. *The Journal of Politics*, 66(1):267–281, 2004. 42

[KSRW04] T. Kohno, A. Stubblefield, A. D. Rubin, and D. S. Wallach. Analysis of an electronic voting system. In *IEEE Symposium on Security and Privacy*, 2004. 2

[KSW05] C. Karlof, N. Sastry, and D. Wagner. Cryptographic voting protocols: A systems perspective. In *USENIX Security Symposium*, 2005. 12, 39

[KT09] R. Kusters and T. Truderung. An epistemic approach to coercion-resistance for electronic voting protocols. In *IEEE Symposium on Security and Privacy*, 2009. 157

[KTV09] R. Kusters, T. Truderung, and A. Vogt. Improving and simplifying a variant of pret a voter. In *VOTE-ID*, 2009. 42

[KTV10a] R. Kusters, T. Truderung, and A. Vogt. Accountability: Definition and relationship to verifiability. In *ACM CCS*, 2010. 12

[KTV10b] R. Kusters, T. Truderung, and A. Vogt. A game-based definition of coercion- resistance and its application. In *CSF*, 2010. 10, 11, 156, 231

[KY02] A. Kiayias and M. Yung. Self-tallying elections and perfect ballot secrecy. In *PKC*, pages 141–158, 2002. 12, 33, 35

[KY04] A. Kiayias and M. Yung. The vector-ballot e-voting approach. In *FC*, 2004. 35

[KY10] A. Kiayias and M. Yung. The vector-ballot approach for online voting procedures. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*, pages 155–174. Springer, 2010. 35

[KZ07] M. Kutylowski and F. Zagorski. Verifiable internet voting solving secure platform problem. In *IWSEC*, 2007. 155, 160

[LBD+03] B. Lee, C. Boyd, E. Dawson, K. Kim, J. Yang, and S. Yoo. Providing receipt-freeness in mixnet-based voting protocols. In *ICISC*, 2003. 34

[Lip10] H. Lipmaa. On the CCA1-security of Elgamal and Damgård's Elgamal. In *Inscrypt*, 2010. 15

[LK00] B. Lee and K. Kim. Receipt-free electronic voting through collaboration of voter and honest verifier. In *JS-ISC*, 2000. 33

[LK02] B. Lee and K. Kim. Receipt-free electronic voting scheme with a tamper-resistant randomizer. In *ICISC*, 2002. 34

[LRSW99] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography*, 1999. 19

[LWL09] F. Liu, C. K. Wu, and X. J. Lin. The alignment problem of visual cryptography schemes. *Designs, Codes and Cryptography*, 50(2), 2009. 140, 141

[MBC01] E. Magkos, M. Burmester, and V. Chrissikopoulos. Receipt-freeness in large- scale elections without untappable channels. In *I3E*, 2001. 33

[MEL86] R. J. Massa, T. R. Ellis, and R. G. LePage. Intelligent surveillance alarm system and method. United States Patent, 4589081, 1986. 163

[Men07] B. Meng. A coercion-resistant internet voting protocol. In *ICSNC*, 2007. 158

[Mer73] R. Merton. Theory of rational option pricing. *Journal of Economics and Management Sciences*, 4(1), 1973. 95

[MH96] M. Michels and P. Horster. Some remarks on a receipt-free and universally verifiable mix-type voting scheme. In *ASIACRYPT*, 1996. 30, 157

[Mil55] G. Miller. Note on the bias of information estimates. In *Information Theory in Psychology II-B*. Free Press, 1955. 102

[MMP02] D. Malkhi, O. Margo, and E. Pavlov. E-voting without 'cryptography'. In *FC*, 2002. 35

[MN06] T. Moran and M. Naor. Receipt-free universally-verifiable voting with everlasting privacy. In *CRYPTO*, 2006. 10, 11, 25, 39

[MN07] T. Moran and M. Naor. Split-ballot voting: Everlasting privacy with distributed trust. In *ACM CCS*, 2007. 73, 75, 77, 231

[MP07] A. Munje and T. Plestid. Password methods and systems for use on a mobile device. United States Patent (Pending), 11181522, 2007. 163

[MPQ07] O. d. Marneffe, O. Pereira, and J.-J. Quisquater. Simulation-based analysis of e2e voting systems. In *VOTE-ID*, 2007. 44, 87

[MS08] T. Moran and G. Segev. David and goliath commitments: Uc computation for asymmetric parties using tamper-proof hardware. In *EUROCRYPT*, 2008. 120

[Nao89] M. Naor. Bit commitment using pseudo-randomness (extended abstract). In *CRYPTO*, pages 128–136, 1989. 124

[Nef04] C. A. Neff. Practical high certainty intent verification for encrypted votes. Technical report, VoteHere Whitepaper, 2004. 23, 33, 39, 69

[NR94] V. Niemi and A. Rendall. How to prevent buying of votes in computer elections. In *ASIACRYPT*, 1994. 29, 35

[NS94] M. Naor and A. Shamir. Visual cryptography. In *EUROCRYPT*, 94. 140, 141

[NSS91] H. Nurmi, A. Salomaa, and L. Santean. Secret ballot elections in computer networks. *Computers & Security*, 36(10):553–560, 1991. 35

[Oht98] K. Ohta. An electrical voting scheme using a single administrator. Technical report, Spring National Convention Record, 1998. 28

[OI10] A. Otsuka and H. Imai. Unconditionally secure electronic voting. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*, pages 107–123. Springer, 2010. 11, 35

[Oka92] T. Okamoto. Provably secure and practical indentification schemes and corresponding signature schemes. In *CRYPTO*, 1992. 20

[Oka96] T. Okamoto. An electronic voting scheme. In *IFIP*, 1996. 31

[Oka97] T. Okamoto. Receipt-free electronic voting schemes for large scale elections. In *Workshop on Security Protocols*, 1997. 10, 11, 32, 222

[OMA+99] M. Ohkubo, F. Miura, M. Abe, A. Fujioka, and T. Okamoto. An improvement on a practical secret voting scheme. In *ISW*, 1999. 31, 36

[Osb03] M. J. Osbourne. *An introduction to game theory*. Oxford University Press, 2003. 74

[OSL08] R. Oppliger, J. Schwenk, and C. Lohr. Captcha-based code voting. In *EVOTE*, 2008. 155

[PAB+04] K. Peng, R. Aditya, C. Boyd, E. Dawson, and B. Lee. Multiplicative homomorphic e-voting. In *INDOCRYPT*, 2004. 33

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999. 17

[Pan03] L. Paninski. Estimation of entropy and mutual information. *Neural Computation*, 15, 2003. 102, 106

[PCE+10] S. Popoveniuc, J. Clark, A. Essex, R. T. Carback, and D. Chaum. Securing optical-scan voting. In *Towards Trustworthy Elections*, volume 6000 of *LNCS*. Springer, 2010. 45

[Ped91] T. P. Pedersen. A threshold cryptosystem without a trusted party. In *EUROCRYPT*, 1991. 16, 123, 180, 181

[Ped92] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1992. 14, 124

[Pet05] R. A. Peters. A secure bulletin board. Master's thesis, Technische Universiteit Eindhoven, 2005. 25

[Pfi94] B. Pfitzmann. Breaking an efficient anonymous channel. In *EUROCRYPT*, 1994. 28, 158

[PH06] S. Popoveniuc and B. Hosp. An introduction to punchscan. In *WOTE*, 2006. 42, 73

[PIK93] C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In *EUROCRYPT*, 1993. 26, 28, 158

[Pop10] S. Popoveniuc. Speakup: remote unsupervised voting. In *ACNS*, 2010. 155

[PR07] M. Prabhakaran and M. Rosulek. Rerandomizable RCCA encryption. In *CRYPTO*, 2007. 17

[Pro04] N. Provos. A virtual honeypot framework. In *USENIX Security Symposium*, 2004. 173

[PW92] B. Pfitzmann and M. Waidner. Unconditionally untraceable and fault-tolerant broadcast and secret ballot election. Technical report, Institut fur Informatik, Universiat Hildesheim, 1992. 11, 35

[Rab83] M. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2), 1983. 91

[RBH+09] P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia. Pret a voter: a voter-verifiable voting system. *IEEE TIFS*, 4(4), 2009. 176

[Res09] E. Rescorla. On the security of election audits with low entropy randomness. In *EVT/WOTE*, 2009. 89, 92, 114

[RS06] P. Y. A. Ryan and S. A. Schneider. Pret a voter with re-encryption mixes. In *ESORICS*, 2006. 42

[RS07] R. L. Rivest and W. D. Smith. Three voting protocols: threeballot, VAV, and twin. In *EVT*, 2007. 43, 44, 87, 144

[RT07] B. Riva and A. Ta-Shma. Bare-handed electronic voting with pre-processing. In *EVT*, 2007. 12, 182

[RT09a] P. Y. A. Ryan and V. Teague. Ballot permutations in pret a voter. In *EVT/WOTE*, 2009. 42

[RT09b] P. Y. A. Ryan and V. Teague. Pretty good democracy. In *Workshop on Security Protocols*, 2009. 154

[Rus05] R. K. Russikoff. Computerized password verification system and method for atm transactions. United States Patent, 6871288, 2005. 163

[Rya04] P. Y. A. Ryan. A variant of the chaum voter-verifiable scheme. Technical Report CS-TR 864, University of Newcastle, 2004. 40

[SCC+10] A. T. Sherman, R. T. Carback, D. Chaum, J. Clark, A. Essex, P. S. Hernson, T. Mayberry, S. Popoveniuc, R. L. Rivest, E. Shen, B. Sinha, and P. L. Vora. Scantegrity mock election at Takoma Park. In *EVOTE*, 2010. 45

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptography*, 4, 1991. 17, 184

[Sch99] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its applications to electronic voting. In *CRYPTO*, 1999. 35, 36

[Sch03] M. Schroder. Brownian excursions and Parisian barrier options: a note. *Advances in Applied Probability*, 40(4), 2003. 97

[Sch06] J. Schweisgut. Coercion-resistant electronic elections with observer. In *EVOTE*, 2006. 158

[SDW08] D. R. Sandler, K. Derr, and D. S. Wallach. VoteBox: a tamper-evident, verifiable electronic voting system. In *USENIX Security Symposium*, 2008. 40, 87, 218

[Sey09] R. U. Seydel. *Tools for computational finance*. Springer, fourth edition, 2009. 95, 96, 99, 101, 105

[SHD10] O. Spycher, R. Haenni, and E. Dubuis. Coercion-resistant hybrid voting systems. In *EVOTE*, 2010. 160

[SK94] K. Sako and J. Kilian. Secure voting using partially compatible homomorphisms. In *CRYPTO*, 1994. 29

[SK95] K. Sako and J. Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In *EUROCRYPT*, pages 393–403, 1995. 25, 28, 30, 36, 59, 157

[SKHS11] O. Spycher, R. Koenig, R. Haenni, and M. Schlapfer. A new approach towards coercion-resistant remote e-voting in linear time. In *FC*, 2011. 158, 178

[Smi05] W. D. Smith. New cryptographic election protocol with best-known theoretical properties. In *Frontiers in Electronic Elections*, 2005. 158, 177

[SR09] M. Smart and E. Ritter. Remote electronic voting with revocable anonymity. In *ISS*, 2009. 161

[Ste08] C. Stewart. Election technology and the voting experience in 2008. Technical report, VTP Working Paper 71, 2008. 1

[TRN08] V. Teague, K. Ramchen, and L. Naish. Coercion resistant tallying for STV voting. In *EVT*, 2008. 36, 231

[TY98] Y. Tsiounis and M. Yung. On the security of ElGamal based encryption. In *PKC*, pages 117–134, 1998. 15, 180

[Tze04] W. G. Tzeng. Efficient 1-out-of-n oblivious transfer schemes with universally usable parameters. *IEEE Transactions on Computers*, 53(2), 2004. 145

[UM10] D. Unruh and J. Muller-Quade. Universally composable incoercibility. In *CRYPTO*, 2010. 231

[Uni09] United States Election Assistance Commission. The 2008 voluntary voting system guidelines, v1.1), 2009. 70

[VG06] M. Volkamer and R. Grimm. Multiple casts in online voting: Analyzing chances. In *EVOTE*, 2006. 160

[WAB07] S. G. Weber, R. S. d. Araujo, and J. Buchmann. On coercion-resistant electronic elections with linear work. In *ARES*, 2007. 158, 177

[WB09a] R. Wen and R. Buckland. Masked ballot voting for receipt-free online elections. In *VOTE-ID*, 2009. 160

[WB09b] R. Wen and R. Buckland. Minimum disclosure counting for the alternative vote. In *VOTE-ID*, 2009. 36

[WJHF04] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for dos resistance. In *ACM CCS*, pages 246–256, 2004. 93

[Wol01] S. Wolfram. *A new kind of science*. Wolfram Media, first edition, 2001. 101

[Wu05] H. Wu. The misuse of RC4 in microsoft word and excel. Cryptology ePrint Archive, Report 2005/007, 2005. 136

[WWB+05] S. Wiedenbeck, J. Waters, J.-C. Birget, A. Brodskiy, and N. Memon. Passpoints: design and longitudinal evaluation of a graphical password system. *Int. J. Hum.-Comput. Stud.*, 63(1-2):102–127, 2005. 173

[XS06] Z. Xia and S. Schneider. A new receipt-free e-voting scheme based on blind signatures. In *WOTE*, 2006. 32

[XSH08] Z. Xia, S. A. Schneider, and J. Heather. Analysis, improvement and simplification of pret a voter with paillier encryption. In *EVT*, 2008. 42

[Yao82] A. C. Yao. Protocols for secure computations. In *IEEE FOCS*, 1982. 11, 35

Appendix A

Optimality Proof for Coercion Contracts

A.1 Algorithm Analysis

Defining Optimality. Recall from Chapter 5 that we say a contract is optimal with respect to Alice if it has the following three properties:

- **O1**: The contract maximizes the expected probability of a vote for Alice (from utility-maximizing voters).
- **O2**: The contract contains no columns where the highest utility is shared between more than one candidate.
- **O3**: The contract contains no columns with all u_0 (a special case of **O2**).

Further recall that we say a contract contains no ambiguity if it meets **O2** and **O3**.

Contract Properties. Recall from Chapter 5 the three properties of a contract with $C \geq 2$:

- **P1**: For each clause with utility u_i in the contract, a column $\hat{k}$ has u_i added to it in the Alice region.
- **P2**: In P1, u_i is always added to the region of each additional candidate in the same row and some column other than $\hat{k}$.
- **P3**: In P2, the (set of) column(s) is either $\{k|\lfloor \frac{k}{C} \rfloor = \lfloor \frac{\hat{k}}{C} \rfloor\}$ or $\{k|k \equiv \hat{k} \mod C\}$. We call these type-A and type-B clauses, respectively.

Algorithm Analysis. Continue to consider the contract as a grid. For ease of exposition, we define the concept of a block within a grid. Block b, $0 \leq b \leq C-1$, is the set of C columns such that $b = \lfloor \frac{k}{C} \rfloor$ for columns $0 < k < C^2 - 1$. Recall from **P3** that a type-A clause is confined to a single block, while a type-B clause adds a u_i to the m^{th} column in each block for $0 \leq m \leq C-1$.

If a column's highest utility is in the Alice region, we way that the contract wins the column. A contract has no advantage over a random vote if it fails to win more than C of the C^2 columns, or has an expected value of $1/C$ with respect to **O1**. The optimal contract generation (OCG) algorithm produces a contract where each column in the first block is won as well as one column in each subsequent block. Thus, it wins $(2C-1)$ of the C^2 columns, has an advantage over a random vote for $C \geq 2$, and, as we now demonstrate, wins the highest number of columns possible for any Punchscan coercion contract.

Toward a contradiction, assume a contract exists that wins W columns where $W > 2C-1$. To show this is impossible, we establish a few more properties:

P4: A non-ambiguous contract constructed from only type-A clauses can never win more than one column per block.
P5: A non-ambiguous contract constructed from only type-A clauses can never win more than C columns in the entire contract.
P6: A non-ambiguous contract constructed from only type-B clauses can never win more than C columns in the entire contract.

Recall from **P3** that a type-A clause adds u_i to each column and, by referencing the definition of a block, that these utilities are within a single block. For a single type-A clause, by **P2**, it will only win a single column. For more than one type-A clause, from **P3**, the clauses will fully overlap each other. This will create ambiguity, contradicting **O2**, if each clause is added with the same u_i (if added with different u_i, the highest clause will win a single column for the same reasoning that a single clause will). Therefore **P4** holds. Since there are C columns, **P5** follows.

Recall from **P3** that a type-B clause adds u_i to a column in each block. For a single type-B clause, by **P2**, it will only win a single column. For more than one type-B clause, from **P3**, the clauses will either fully overlap each other or not overlap at all. By the same reasoning for overlapping type-A clauses, a non-ambiguous contract can only win a single column per non-overlapping type-B clause, even with the use of different levels of u_i. Since, from **P3**, there are C non-overlapping clauses of type-B, **P6** holds. We now establish another property:

P7: A non-ambiguous contract constructed from type-A *and* type-B clauses can never win more than $2C$ columns.

P7 follows directly from applying the triangle inequality to **P5** and **P6**. It serves as an upper-bound on the number of columns a contract could win, and differs only by one column from what we claim is actually achievable. Finally, we tighten the bound to the exact result:

P8: A non-ambiguous contract constructed from type-A and type-B clauses can never win more than $2C - 1$ columns.

Given **P7** and the fact that OCG wins $2C - 1$ columns, there are only two options for an optimal contract: either it wins exactly $2C$ columns or it is equivalent to OCG, winning $2C - 1$. Thus it only remains to be shown that winning exactly $2C$ columns is impossible.

A contract consists of C blocks with C columns each. Given **P5** and **P6**, the only way a contract can win $2C$ columns is with C winning type-A clauses and C winning type-B clauses. Since winning type-A clauses cannot fully overlap (as mentioned above; from **P3**), each block must have exactly one type-A clause that wins a column in the block.

To add a type-B clause that wins a column in a block, it must deposit a utility greater than the type-A clause in the same block. If there are C winning type-B clauses, they must all have utilitiy greater than the respective type-A clauses. Since winning clauses do not overlap, across the C type-B clauses, a utility is added to each column. This implies a contradiction. One on hand, there must be at least one type-A clause that has a lower utility than all type-B clauses if all type-B clauses are winning. One the other hand, there must be a winning type-A clause in each block, which means a higher utility than the type-B clause competing for its column. (Note that we could do this proof by starting with C type-B clauses and show that adding C type-A clauses also leads to a contradiction.) Therefore, by contradiction, **P8** holds.

Given **P8**, no contract can better maximize the expectation of votes for Alice than a contract generated with OCG. Thus, OCG is optimal with respect to **O1**. Further it should be clear from inspection (and extrapolating Figure 5.4) that an OCG contract contains no ambiguity, thereby ensuring **O2** and **O3**. Given the OCG algorithm satisfies **O1**, **O2**, and **O3**, we say the OCG algorithm is optimal. □

Intuition. For further intuition, consider the OCG algorithm (see Figure 5.4 in Chapter 5). In the resulting contract for any $C \geq 2$, all C winning type-B clauses are packed into the first block (with u_1). This means no type-A clause can win the first block (or equivalently, the first block contains a type-A clause of u_0 which is less than all type-B clauses). However type-A clauses can be used to win the remaining $C - 1$ blocks (with u_2). A corollary of this result is that optimality can be achieved with only three levels of utility.

Appendix B

Security Proof for the Random Beacons

B.1 Security Analysis

Let n, m, and l be positive integers such that $n \geq 2m \geq 4l$. Recall the main operation of our protocol:

$$s_i = \texttt{Ext}'_{k'}(\texttt{Eval}_{sk}((s_{i-1} \| s_{i-2}) \oplus \texttt{Ext}_k(\mathcal{P}_i))). \tag{B.1}$$

It is used for updating seeds $s_i \in \{0,1\}^l$. The stock market randomness, $\mathcal{P}_i$, is n bits (broken into L blocks of m bits). The operation uses two extractors with different output sizes: `Ext` has an m-bit output and uses an extraction key $k \in \{0,1\}^m$, while `Ext'` has an l-bit output and is keyed with $k' \in \{0,1\}^l$. The BSP's secret key $sk \in \mathbb{Z}_q^*$, where q is a large prime of at least m bits. For reference, we might consider $l = 128$, $m = 256$, and q as a prime of at least 256 bits. In this section, we show our protocol has the following properties:

- **Verifiable.** A polynomial time verifier should be convinced that `Update` was computed from $\mathcal{P}_i$, s_{i-1}, and s_{i-2} as specified in the protocol.
- **Statistically Random.** The statistical difference between s_i and a uniform distribution of the same length should be ϵ-close.
- **Unpredictable.** The advantage of a PPT-bound adversary in predicting s_i without $\mathcal{P}_i$ is negligible, even if the adversary knows the secret key, s_{i-1} and s_{i-2}.
- **Parisian.** The advantage of a PPT-bound adversary in computing the exact value of $\mathcal{P}_i$ needed to output a chosen s_i is negligible if s_i is chosen before knowing $\mathcal{P}_{i-1}$ and s_{i-1} and s_{i-2}, even if she knows the secret key.

- **Partially Distributed.** The advantage of a PPT-bound adversary in predicting s_i is negligible if she knows s_{i-1} and s_{i-2}, chooses $\mathcal{P}_i$, but does not know the BSP's secret key.

B.1.1 Key Derivation

Before showing that our protocol meets the described properties, we describe a procedure for deriving two extractor keys: $k \in \{0,1\}^m$ and $k' \in \{0,1\}^l$. For simplicity, consider this the task of generating a single key of size $\kappa = m + l$ and then partitioning it into two keys. Recall that unlike cryptographic keys, extraction "keys" do not need to be unpredictable or kept secret. The key is an index to each extractor in a family of extractors. The key does need to be selected with uniform randomness to ensure there is no bias in choosing an extractor from the family.

To generate a uniformly random key of length κ, we will use historic prices of a single stock. The choice of stock is arbitrary but for consistency, we can use the S&P 500 aggregate index, which produces a single representative price for the 500 stocks contained in the portfolio. Instead of using the prices themselves, we will use the price differences between two consecutive days beginning at some agreed upon date. This will give us a list of price movements: up, down, or (rarely) no change. From this list, we apply the Von Neumann extractor for a biased coin. We consider up to be heads, down to be tails, and in cases of no change, an output bit will not be generated.

To do this, we partition the list into pairs of values. (The importance of agreeing on a beginning date is only to ensure this partition has the same alignment when a verifier duplicates the procedure.) If the pair is {down,up}, we output a 0. If the pair is {up,down}, we output a 1. If the output is anything else (*i.e.,* {down,down}, {up,up}, or involves a no change), we output no value.

Of the Black-Scholes assumptions, this extraction is only assuming that for all (non-overlapping) intervals in time, the price difference is statistically independent from all other such intervals—*i.e.,* it is a Markov process. It does not matter if there is drift or diffusion, or even if these factors change in magnitude over time. In the same way that the Von Neumann extractor works even if the coin is biased, drift affects the result in the same way: the more bias, the less the number of random bits that can be extracted on average.

For our update function with the reference parameters, $\kappa = 256 + 128 = 384$ bits. If we start our process on January 1, 2004, the S&P 500 generates over 420 bits by March 23, 2010. We use these bits, ordered from oldest to newest to construct a bitstring from most-significant bits to least-significant bits respectively. If the bitstring is longer than the required key, the rightmost bits are used. Each day, the BSP checks the latest price movement. If a new bit is generated, it can be shifted into the least-significant bit of the key

and the most-significant is dropped. Although extraction keys do not need to be refreshed, this can help verifiers regenerate the key without going as deep into historic prices. Since the keys are evolving, we herein denote their value at time i as k_i and k'_i.

B.1.2 Verifiable

We have already shown, from Algorithm 4, that the protocol is verifiable. It is easy to see that aside from $\texttt{Eval}_{sk}$, the protocol is completely deterministic with knowledge of s_{i-1}, s_{i-2}, and $\mathcal{P}_i$. Dodis and Yampolskiy show that for the specific $\texttt{Eval}_{sk}$ we implement, one can verify that known output is the result of the function being evaluated on some known input [DY05]. This verification has already been summarized in Section 6.5.1.

B.1.3 Statistically Random

We consider whether the output is *statistically random*. This is not a security property: an output can have a random distribution while specific outputs are predictable to an adversary. In our case, we want the output to be high entropy in both a statistical sense, and from a defined view by the adversary of the protocol. However in certain circumstances, for example common reference strings or sunspots, unpredictability is not necessary and so we treat the issues separately. Another example that we have encountered where randomness is needed, but not unpredictability, is an extractor key.

We say an update function is *statistically random* if the statistical distance between the produced seed and a uniformly random seed is negligible in some security parameter.

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be discrete distributions. Let Z be the set of events in the distribution, and $\Pr_{\mathcal{D}}(z)$ be the probability that event $z \in Z$ occurs under distribution $\mathcal{D}$. Recall [DGH+04] that the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is:

$$\mathsf{SD}(\mathcal{D}_1, \mathcal{D}_2) = \frac{1}{2} \sum_{z \in Z} |\Pr_{\mathcal{D}_1}(z) - \Pr_{\mathcal{D}_2}(z)|.$$

When $\mathcal{D}_2$ is a uniform distribution over Z, $\mathcal{D}_2 = \mathbf{U}_{|Z|}$, then the statistical distance is:

$$\mathsf{SD}(\mathcal{D}_1, \mathcal{D}_2) \leq \frac{1}{2} \sqrt{|Z| \cdot 2^{H_\infty(\mathcal{D}_1)} - 1}. \tag{B.2}$$

We first apply a result from Dodis *et al.* concerning CBC-MAC mode extractors [DGH+04]. Recall that $\texttt{Ext}_{k_i}$ (the inner extractor in our update function) denotes the

CBC-MAC mode over a random permutation. $\texttt{Ext}_{k_i}$ takes an input of arbitrary length and m-bit uniform random key, and produces an m-bit output. Let $\mathcal{P}_i$ denote an n-bit representation of stock prices (over L blocks of length m) and $\mathbf{P}_n$ be the distribution of possible $\mathcal{P}_i$ values over $\{0,1\}^n$. Let $\mathbf{U}_m$ be a random variable with uniform distribution over $\{0,1\}^m$. Provided $\mathbf{P}_n$ has min-entropy $H_\infty(\mathbf{P}_n) > 2m$ and its length, in blocks, is restricted by $L < 2^{m/4}$, the statistical distance between $\texttt{Ext}_{k_i}(\mathbf{P}_n)$ and $\mathbf{U}_m$ is:

$$\mathsf{SD}(\texttt{Ext}_{k_i}(\mathbf{P}_n), \mathbf{U}_m) \leq \mathcal{O}(L \cdot 2^{-\sqrt{m}}) = \mathsf{negl}(m).$$

Reconsider this step with the reference parameters. We have conjectured that the S&P 500 has at least 512 bits of min-entropy, which allows $m = 256$. The largest price in the last decade of the S&P was reached by GOOG at \$741.79. As integer 74 179, it (barely) requires 17 bits. If we represent 500 prices at 17 bits each, $\mathcal{P}_i$ will require $L = 34$ blocks and $L < 2^{256/4}$. Therefore, the conditions for the stated result are met and the statistical distance is negligible in m.

Let $\mathbf{X}_m$ be the distribution over the range of $(s_{i-1} \| s_{i-2}) \oplus \texttt{Ext}_{k_i}(\mathbf{P}_n)$ with domain distribution $\mathbf{P}_n$. The entropy in $\texttt{Ext}_{k_i}(\mathbf{P}_n)$ is used to mask $(s_{i-1} \| s_{i-2})$, as in a one-time pad, and therefore s_{i-1} and s_{i-2} can be considered fixed constants. Thus the min-entropy of $\texttt{Ext}_{k_i}(\mathbf{P}_n)$ is preserved in $\mathbf{X}_m$ and

$$\mathsf{SD}(\mathbf{X}_m, \texttt{Ext}_{k_i}(\mathbf{P}_n)) = 0.$$

We now recall a result from Dodis and Yampolskiy concerning their verifiable unpredictable function, $y_i = \texttt{Eval}_{sk}(x_i) = g^{\frac{1}{x_i + sk}}$ and $\texttt{Verify}_{pk}(x_i, y_i, \pi_i) : e(pk \cdot g^{x_i}, y_i) \stackrel{?}{=} e(g, g)$ [DY05]. A VUF demonstrates uniqueness if $\texttt{Verify}_{pk}(x, y_1, \pi_1)$ and $\texttt{Verify}_{pk}(x, y_2, \pi_2)$ cannot hold for the same x, same pk (and thus same sk), and $y_1 \neq y_2$. For this particular VUF, where π is encapsulated in y, uniqueness implies $x \rightarrow \texttt{Eval}_{sk}(x)$ is an injective function.

Injective functions are entropy-preserving: thus y_i preserves the entropy of x_i. Let $\mathbf{Y}$ be the distribution over the range of $\texttt{Eval}_{sk}(\mathbf{X}_m)$ with domain distribution $\mathbf{X}_m$. $\mathbf{Y}$ will consist of points on some elliptic curve, so we do not denote its length. Because $\texttt{Eval}_{sk}$ is injective for a fixed sk, the min-entropy of $\mathbf{Y}$ will be equivalent to $H_\infty(\mathbf{X}_m)$. Thus:

$$\mathsf{SD}(\mathbf{Y}, \texttt{Eval}_{sk}(\mathbf{X}_m)) = 0.$$

Let $\mathbf{U}_{\hat{m}}$ be a uniform distribution over the support of $\texttt{Eval}_{sk}(\mathbf{U}_m)$. We use $\hat{m}$ for preciseness because while the distribution is define over a set of the same size as $\{0,1\}^m$,

the symbols are not m-bit bitstrings but rather coordinates on an elliptic curve. Putting the above steps together, we have that:

$$\mathsf{SD}(\mathtt{Eval}_{sk}((s_{i-1}\|s_{i-2}) \oplus \mathtt{Ext}_{k_i}(\mathbf{P}_n)), \mathbf{U}_{\hat{m}}) = \mathsf{negl}(m).$$

The reason for this analysis is to establish the min-entropy of $\mathbf{Y}$ in preparation for the outer extraction $\mathtt{Ext}'_{k'_i}$.[1] For the inner extraction we just analyzed, we required the min-entropy to be twice the output length. For the outer extractor, $\mathtt{Ext}'_{k'_i}$, the output is l where $m = 2l$. Therefore to analyze it in the same way, the input should have at least m bits. However since the statistical distance between $\mathbf{Y}$ and $\mathbf{U}_{\hat{m}}$ is not exactly 0, the min-entropy of $\mathbf{Y}$ is less than m bits.

Let $\delta = |H_\infty(\mathbf{Y}) - H_\infty(\mathbf{U}_{\hat{m}})|$. Let $\mathbf{U}_l$ be a random variable with uniform distribution over $\{0,1\}^l$. The statistical distance can be determined (using bounds from Dodis *et al.* [DGH+04] when the number of blocks in the input is $L' < 2^{k/4}$) as:

$$\begin{aligned} & \mathsf{SD}(\mathtt{Ext}'_{k'_i}(\mathbf{Y}), \mathbf{U}_l) \\ \leq\ & \sqrt{2^l 2^{-H_\infty(\mathbf{Y})} + \mathcal{O}(4L'^2 2^{-l} + L'^6 2^{-2l})} \\ \leq\ & \mathcal{O}\left(\sqrt{2^{-l}(L'^2 + 2^{-\delta\ l})}\right) \\ =\ & \mathsf{negl}(l). \end{aligned} \tag{B.3}$$

Note that when $\delta = 0$, the previous bound of $\mathcal{O}(L' \cdot 2^{-\sqrt{l}})$ holds. With the reference parameters, assume that Y is a pair of coordinates on an elliptic curve defined over a finite field. We assumed the order of the subgroup q was a prime of at least 256 bits and now let us assume the base field size is, say, 1024 bits. With a Y of 2048 bits, we can extract $l = 128$ bits. The input will be $L' = 16$ blocks satisfying $L' < 2^{128/4}$.

We say the update function in Equation B.1 is *statistically random* because as shown in Equation B.3, the statistical distance between the produced seed and a uniformly random seed is negligible in l.

B.1.4 Unpredictable

Assume an adversary knows sk. We say an update function is *unpredictable* if a PPT-bounded adversary cannot distinguish between the following two distributions:

[1] Note we can relate statistical distance and min-entropy through Equation B.2 or the more specific bounds we cite from [DGH+04].

$$
\begin{aligned}
s_i &= \mathtt{Ext}'_{k'_i}(\mathtt{Eval}_{sk}((s_{i-1}\|s_{i-2}) \oplus \mathtt{Ext}_{k_i}(\mathbf{P}_n))) \\
s_i &= \mathbf{U}_l
\end{aligned}
$$

We show this in the context of the following security game: an oracle returns to the adversary a value of s_i drawn from one of the two distributions with equal probability. The adversary's goal is to guess which was sent. We will show the adversary's advantage at winning this game over a random guess is negligible through a series of related games.

Recall $\mathbf{U}_m$ is a random variable with uniform distribution over $\{0,1\}^m$. For any $\mathcal{P}_i$ selected from the distribution of closing prices, a PPT-bounded adversary (in m and l) cannot distinguish, with non-negligible advantage, between the following two statements:

$$
\begin{aligned}
s_i &= \mathtt{Ext}'_{k'_i}(\mathtt{Eval}_{sk}((s_{i-1}\|s_{i-2}) \oplus \mathtt{Ext}_{k_i}(\mathcal{P}_i))) \\
s_i &= \mathtt{Ext}'_{k'_i}(\mathtt{Eval}_{sk}((s_{i-1}\|s_{i-2}) \oplus \mathbf{U}_m)).
\end{aligned}
$$

This is because the statistical distance, as shown in the previous section, is negligible in m and the adversary only has polynomial capabilities relative to m. Since $(\mathtt{Eval}_{sk}((s_{i-1}\|s_{i-2}) \oplus \mathbf{U}_m)$ acts as a one-time pad, any adversary (bounded or not), cannot distinguish:

$$
\begin{aligned}
s_i &= \mathtt{Ext}'_{k'_i}(\mathtt{Eval}_{sk}((s_{i-1}\|s_{i-2}) \oplus \mathbf{U}_m)) \\
s_i &= \mathtt{Ext}'_{k'_i}(\mathtt{Eval}_{sk}(\mathbf{U}_m)).
\end{aligned}
$$

We showed in the previous section that $y_i = \mathtt{Eval}_{sk}(x_i)$ is injective. Thus, any adversary (bounded or not), cannot distinguish:

$$
\begin{aligned}
s_i &= \mathtt{Ext}'_{k'_i}(\mathtt{Eval}_{sk}(\mathbf{U}_m)) \\
s_i &= \mathtt{Ext}'_{k'_i}(\mathbf{U}_{\hat{m}}).
\end{aligned}
$$

Recall that $\mathbf{U}_l$ is a random variable with uniform distribution over $\{0,1\}^l$. A PPT-bounded adversary cannot distinguish, with non-negligible advantage, between the following two statements:

$$
\begin{aligned}
s_i &= \mathtt{Ext}'_{k'_i}(\mathbf{U}_{\hat{m}}) \\
s_i &= \mathbf{U}_l.
\end{aligned}
$$

This is because the statistical distance, as shown in the previous section, is negligible in l. Putting this chain of modifications together, an adversary's advantage in distinguishing the following is negligible:

$$
\begin{aligned}
s_i &= \mathtt{Ext}'_{k'_i}(\mathtt{Eval}_{sk}(\mathtt{Eval}_{sk}((s_{i-1} \| s_{i-2}) \oplus \mathtt{Ext}_{k_i}(\mathcal{P}_i))) \\
s_i &= \mathbf{U}_l.
\end{aligned}
$$

More precisely, it is: $\epsilon = \text{negl}(m) + 0 + 0 + \text{negl}(l)$, where $m = 2l$. Therefore, the update function in Equation B.1 is unpredictable with respect to l.

B.1.5 Parisian

Assume an adversary knows sk. We say an update function is *Parisian* if an adversary, given a target value for seed s_i, cannot compute the value of a valid $\mathcal{P}_i$ such that the update function produces s_i.

We show this through a two-step reduction: reductions from the hardness of finding x_i in the first equation to the hardness of finding x_i in the second to the hardness of finding $\mathcal{P}_i$ in the third.

$$
\begin{aligned}
y_i &= g^{x_i} \\
y_i &= \mathtt{Eval}_{sk}(x_i) \\
s_i &= \mathtt{Ext}'_{k'_i}(\mathtt{Eval}_{sk}((s_{i-1} \| s_{i-2}) \oplus \mathtt{Ext}_{k_i}(\mathcal{P}_i)))
\end{aligned}
$$

Toward a contradiction, assume there exists an adversary $\mathcal{A}_1$ who can efficiently find x_i, given g, sk, and $y_i = \mathtt{Eval}_{sk}(x_i) = g^{\frac{1}{sk+x_i}}$ for some group $\mathbb{G}_q$ where the discrete logarithm problem is hard. In other words, there is an oracle for the second line in our reduction steps. Such an adversary could find an arbitrary discrete logarithm in $\mathbb{G}_q$, say $log_\beta(\alpha)$, by setting $y = \beta$, $g = \alpha$, choosing a random sk and finding x_i. It then returns $log_\beta(\alpha) = x_i + sk$. Therefore $\mathcal{A}_1$ can only exist if there is an efficient adversary that can solve arbitrary discrete logarithms in $\mathbb{G}_q$.

Now assume there exists an efficient adversary $\mathcal{A}_2$ who can efficiently find $\mathcal{P}_i$, given sk, s_{i-1}, k_i, k'_i, and $s_i = \mathtt{Ext}'_{k'_i}(\mathtt{Eval}_{sk}((s_{i-1}\|s_{i-2}) \oplus \mathtt{Ext}_{k_i}(\mathcal{P}_i)))$—an oracle for line 3. Such an adversary could solve the previous problem of finding arbitrary pre-images to $\mathtt{Eval}$. Say for example, the adversary is challenged to find $\delta = \mathtt{Eval}_{sk}(\gamma)$. It can choose a random sk, k_i, k'_i, and s_{i-1}. It sets $s_i = \mathtt{Ext}'_{k'_i}(\delta)$, finds $\mathcal{P}_i$, and returns $\gamma = s_{i-1} \oplus \mathtt{Ext}_{k_i}(\mathcal{P}_i)$. Therefore $\mathcal{A}_2$ only exists if $\mathcal{A}_1$ exists, and $\mathcal{A}_1$ only exists if the discrete logarithm problem is easy in $\mathbb{G}_q$. Assuming, as is standard, that the discrete logarithm problem is hard in $\mathbb{G}_q$, $\mathcal{A}_2$ does not exist and Equation B.1 is parisian.

B.1.6 Partially Distributed

For the final property, assume an adversary does not know sk. We say an update function is *partially distributed* if the value of s_i cannot be computed by the adversary given all other inputs.

We show this property by showing that s_i is not computable even if the adversary knows every input, except sk, to,

$$s_i \quad = \quad \mathtt{Ext}'_{k'_i}(\mathtt{Eval}_{sk}((s_{i-1}\|s_{i-2}) \oplus \mathtt{Ext}_{k_i}(\mathcal{P}_i))).$$

Computing s_i requires computing $\mathtt{Eval}_{sk}$ on a known input, without knowledge of sk. Dodis and Yampolsky give a reduction to a bilinear decisional problem, q-DDHI, for the difficulty of this task (which they call unpredictability) for their their VUF [DY05], but only when the input is superlogirithmic. An alternative approach for the same VUF is undertaken by Camenisch *et al.* [CHK$^+$06]. They analyze the VUF in the generic group model and find with this stronger assumption, it can admit full length inputs. For our beacon protocol, we require full length inputs and thus rely on this weaker result.

Appendix C

Security Proof for Eperio

C.1 Security Proof

In this section, we prove that Eperio is *complete*, *sound*, and *computationally secret*. Given a transcript of the entire protocol, the asserted tally can be either accepted or rejected. If the transcript is correct and matches the asserted tally, the decision will always be to accept (completeness). If the asserted tally is not correct, the decision will be to reject with a high probability (soundness). Finally, the outputs do not provide any information that can be used by a computationally bounded adversary to determine any non-negligible information about which candidate was voted for by any voter (computational secrecy).

To maintain consistency with the literature, we will denote the trustees as the prover ($\mathcal{P}$), who we initially consider unbounded in computation power. The three types of verifiers ($\mathcal{V}$) are considered a single entity and to be bounded to probabilistic polynomial time (PPT). These bounds are used to demonstrate unconditional integrity, and are later reversed for a brief consideration of everlasting privacy. Either entity may employ a malicious strategy and we denote this with a prime ($\mathcal{P}', \mathcal{V}'$).

The Eperio verification protocol is challenge-response. The print audit uses an *interactive challenge* where verifiers choose which ballots at random to check that the correspondence between ballot positions and candidates is consistent with the committed $\mathbf{E}$ table. The receipt check also uses an *interactive challenge* where a random fraction of voters will choose to check that the correspondence between ballot positions and marks is consistent with the published marks $\mathbf{E}_{(:,2,:)}$. Finally the tally audit uses a *public coin challenge* where the prover is asked to reveal either $\mathbf{E}_{(:,1,x)}$ or $\mathbf{E}_{(:,3,x)}$ for a given x.

Let Ω represent the description of an election; a mapping between each ballot and the candidate who was voted. Consider the pair $\langle \mathbf{E}, \mathbf{R} \rangle$, an Eperio table and the random

factors used in committing to it. These two items are sufficient for generating all the other data structures needed in the verification protocol. Denote $\mathcal{E}$ as the set of all $\langle \mathbf{E}, \mathbf{R} \rangle$ pairs that would correctly encode Ω, and $\bar{\mathcal{E}}$ as the set of all other pairs.

C.1.1 Completeness

An abbreviated transcript of the verification protocol follows:

$\mathcal{P} \rightarrow \mathcal{V}$:	$\mathbf{C}$	Commitment
$\mathcal{P} \leftarrow \mathcal{V}$:	$\mathbf{M}$	Challenge
$\mathcal{P} \rightarrow \mathcal{V}$:	ρ, $\mathbf{L}$, $\mathbf{E}_{(:,2,:)}$, τ	Response
$\mathcal{P} \leftarrow$ `Coin`:	$\mathbf{Z}$	Challenge
$\mathcal{P} \rightarrow \mathcal{V}$:	$\mathbf{R}_z$	Reveal

Recall that τ represents the asserted tally and let ρ be the asserted receipts. Assume $\mathbf{E} \in \mathcal{E}$ for Ω. If $\mathcal{P}$ and $\mathcal{V}$ behave honestly and follow the protocol outlined in the previous section, then τ, ρ and $\mathbf{L}$ will also be consistent with Ω and the three verifications performed by $\mathcal{V}$ will succeed with certainty.

It may be worthwhile to note that $\mathcal{V}$'s acceptance is conditional on many things beyond τ. If $\mathcal{P}$ were to commit to wrong values and reveal marks in the wrong position, but still report the correct τ, the proof will be rejected. We do not classify this as a true negative error as we are determining whether the proof is acceptable.

C.1.2 Soundness

We now consider whether $\mathcal{V}$ will ever accept a proof when $\mathbf{E} \in \bar{\mathcal{E}}$. If the probability of rejecting a false proof is overwhelming, we say Eperio is sound. The soundness of Eperio relies on two assumptions:

1. The function $\mathsf{Commit}(m, r)$ is perfectly binding. That is, for any m_1 such that $\mathsf{Commit}(m_1, r_1) = c_1$, there does not exist any r_2 and $m_2 \neq m_1$ such that $\mathsf{Reveal}(c_1, r_2) = m_2$.
2. The function $\mathsf{PublicCoin}()$ is perfectly unpredictable before invocation.

Let $\mathrm{tr}_{(\mathcal{P}'(\mathbf{E},\Omega),\mathcal{V}(\mathbf{M},\mathbf{Z}))}(\alpha)$ denote a transcript of a proof given by a malicious prover with private information $\mathbf{E}_{(i,j,k)}$ and Ω to an honest verifier with challenges $\mathbf{M}$ and $\mathbf{Z}$. The proof concerns some public information α. We shorten the notation to $\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\alpha)$. Given a transcript, $\mathcal{V}$ will decide in PPT-time to either `ACCEPT` or `REJECT` the transcript as a valid proof.

The logic of $\mathcal{V}$'s strategy in terms of the three types of verification is as follows,

$$\begin{aligned}\texttt{ACCEPT}_{\mathcal{V}}(\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\mathbf{C},\mathbf{L},\rho,\tau,\mathbf{E}_{(:,2,:)},\mathbf{R}_z)) \quad &\leftarrow \\ \texttt{ACCEPT}_{\mathcal{V}_1}(\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\rho)) \quad &\wedge \\ \texttt{ACCEPT}_{\mathcal{V}_2}(\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\mathbf{C},\mathbf{L},\mathbf{R}_z)) \quad &\wedge \\ \texttt{ACCEPT}_{\mathcal{V}_3}(\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\mathbf{C},\rho,\tau,\mathbf{E}_{(:,2,:)},\mathbf{R}_z)). \quad &\text{(C.1)}\end{aligned}$$

We prove the soundness of the three right-hand decisions independently and then consider the implication of their conjunction. Since the proof protects against very strong adversaries, we also remark on what a real world adversary may want to achieve as it relates to each proof.

Receipt Check. First consider the verification that the asserted receipts, ρ, match the physical receipts retained by the voters: $\texttt{ACCEPT}_{\mathcal{V}_1}(\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\rho))$. This check does not prove anything about the committed values directly, but is simply designed to decide whether ρ alone is reputable. If it is, this fact can be used to establish other facts more directly associated with the asserted tally τ.

A malicious prover has a binary choice at this stage: to assert a correct value ρ or an incorrect ρ'. If ρ' is asserted, the transcript's acceptability decreases in proportion to the number of voters verifying their receipts. Let b'_r be the number of modified ballot receipts in ρ', and $0 \leq p_1 \leq 1$ represent the fraction of voters who conduct a receipt check. Further assume the distribution of b'_r and the ballots associated with p_1 are independent. In this case, the probability of rejecting an incorrect transcript is tightly approximated by:

$$\mathbf{Pr}[\texttt{REJECT}_{\mathcal{P}',\mathcal{V}_1}] \quad = \quad (1-(1-p_1)^{b'_r}). \qquad \text{(C.2)}$$

In terms of real-world adversaries, this verification step is very important. A malicious set of trustees could use such an attack to explicitly move votes from one candidate to another in ρ. This result can be generalized to any voting system where voters check the inclusion of their obfuscated votes. Increasing certainty in the tally proof can often be made exponential in some chosen parameter (as we will show is the case with Eperio), but this certainty will at some point overtake the certainty that the inputs are correct, which are fixed by p_1 and invariant to the security parameter of the system. Since the soundness of the overall system rests in the weakest link—increasing the certainty of the tally does not necessarily add security overall! Further attention should be paid to methods for increasing partition in the receipt check, or introducing mechanisms to capture all the receipts. The only system we are aware of to address this is VoteBox, which includes their earlier AUDTITORUM protocol that could capture all receipt information [SDW08].

Print Audit. Next consider $\mathrm{tr}_{(\mathcal{P}',\mathcal{V})}(\mathbf{C}, \mathbf{L}, \mathbf{R}_z)$ which is used to determine if the linkage list, $\mathbf{L}$, implies that the ballots were printed in a way that is consistent with the binary relation $\mathrm{R}(\mathbf{E}_{(i,1,k)}, \mathbf{E}_{(i,3,k)})$ for all i and k. These values are committed to in $\mathbf{C}$ and the partial relation $\mathbf{E}_{(i,2z+1,k)}$ is opened by $\mathbf{R}_z$.

If a malicious prover misprints b'_p ballots, and $0 \leq p_2 \leq 1$ is the fraction of ballots that are print audited, the probability of a modified ballot being selected for audit is $1-(1-p_2)^{b'_p}$. There are different ways to misprint a ballot, however any surjective modifications (*e.g.,* printing the same candidate twice) are trivially detectable. Consider a misprinted ballot to be one that bijectively swaps the order of some candidates.

$\mathcal{P}'$ could escape detection by decommiting to the misprinted value, however we assume this is impossible by soundness assumption 1. Instead, $\mathcal{P}'$ must assert which i in $\mathbf{E}_{(i,j,k)}$ matches the printed ballot. For a given k, $\mathcal{P}'$ could choose i such that $\mathbf{E}_{(i,1,k)}$ is correct, and will escape detection if the coin flip is 0 but be detected if $z = 1$. Alternatively, the prover could choose the row such that $\mathbf{E}_{(i,3,k)}$ is correct, and will escape detection if $z = 1$ but be detected if $z = 0$. Given the second soundness assumption, $\mathcal{P}'$ cannot predict z to a probability better than $1/2$ for each instance $1 \leq k \leq x$. In this case, the probability of rejecting an incorrect transcript is approximated by:

$$\mathbf{Pr}[\mathtt{REJECT}_{\mathcal{P}',\mathcal{V}_2}] \;=\; (1-(1-p_2)^{b'_p})(1-\frac{1}{2^x}). \tag{C.3}$$

For small numbers, it may be better to explicitly state the number of ballots printed b_p and the number of ballots audited b_a than to express them as a fraction p_2. In this case, the exact probability is:

$$\begin{aligned}\mathbf{Pr}[\mathtt{REJECT}_{\mathcal{P}',\mathcal{V}_2}] &= \left(1-\frac{\binom{b_p-b'_p}{b_a}}{\binom{b_p}{b_a}}\right)(1-\frac{1}{2^x}) \\ &= \frac{(b_p-b'_p)!(b_p-b_a)!}{b_p!(b_p-b'_p-b_a)!}(1-\frac{1}{2^x}).\end{aligned} \tag{C.4}$$

In our estimate, a real-world adversary is unlikely to gain from misprinting ballots. The attack would begin with some prior knowledge of the expected distribution of votes—*e.g.*, pre-election polls showing another candidate leading or an expected bias for a given precinct—and would involve mapping the leading candidate's votes to the adversary's preferred candidate and the least preferred candidate to the main competitor. With $x = 20$ and 5% of ballots being print audited, the election authority can be 99% certain that no more than 89 ballots were misprinted and 50% certain that no more than 13 were. With

20% checked, no more than 20 ballots were misprinted with 99% certainty and no more than 3 with 50% certainty. While a 99% detection rate tolerates a fairly high number of misprinted ballots, this attack is marginalized by the observation that a successful execution will only probabilistically gain votes for the attacker. For example, if working from poll data showing a 51-49% split between two candidates, misprinting 89 ballots by swapping the candidates' positions on the ballot will on average only net the attacker 2 votes.[1]

Tally Audit. Finally consider the verification that the now validated receipt list, ρ, implies that the asserted tally, τ, is correct. The partial relation $\mathbf{E}_{(i,2z+1,k)}$ is opened by $\mathbf{R}_{(z+1,j)}$ and can be considered along side $\mathbf{E}_{(:,2,:)}$). If $z = 0$, this pair of values should match ρ. If $z = 1$, the pair should match τ. Thus our consideration is
$\texttt{ACCEPT}_{\mathcal{V}_3}(\text{tr}_{(\mathcal{P}',\mathcal{V})}(\mathbf{C}_{(i,j)}, \rho, \tau, \mathbf{E}_{(:,2,:)}, \mathbf{R}))$.

If $\mathbf{E}_{(:,2,:)}$ is correct, then the tally produced by any $\mathbf{E}_{(i,3,k)}$ is also correct. However if $\mathbf{E}_{(:,2,:)}$ is not correct, it is possible for $\mathcal{P}'$ to assert a tally τ' that is incorrect and acceptable to $\mathcal{V}$. For a given k, $\mathcal{P}'$ could choose $\mathbf{E}_{(:,2,:)}$ to match τ, and will escape detection if the coin flip is 0 but be detected if $z = 1$. Alternatively, $\mathcal{P}'$ could choose $\mathbf{E}_{(:,2,:)}$ to match τ', and will escape detection if $z = 1$ but be detected if $z = 0$. Given the second soundness assumption, $\mathcal{P}'$ cannot predict z to a probability better than $1/2$ for each instance $1 \leq k \leq x$. In this case,

$$\mathbf{Pr}[\texttt{REJECT}_{\mathcal{P}',\mathcal{V}_3}] \quad = \quad (1 - \frac{1}{2^x}). \tag{C.5}$$

In terms of real-world adversaries, this verification step is very important. Like in the receipt check, a malicious set of trustees could use such an attack to explicitly move votes from one candidate to another in τ.

Overall Effective Soundness. We assume $\mathcal{P}'$ will adopt a proof strategy that minimizes the soundness of the proof. Thus given a transcript with an incorrect public input, the probability of it being rejected by V is,

$$\mathbf{Pr}[\texttt{REJECT}_{\mathcal{P}',\mathcal{V}}] = \min[(1 - (1 - p_1)^{b'_r}), (1 - (1 - p_2)^{b'_p})(1 - \frac{1}{2^x}), (1 - \frac{1}{2^x})]. \tag{C.6}$$

By estimating p_1 and b'_r, we can use this equation to determine a suitable x for our implementation. For example, if the close 2008 Minnesota senate race was run with Eperio, the attacker would need to modify at least 113 marks to overthrow the 225 victory margin.[2]

[1] This assumes the misprinted ballots are distributed to a random sample of voters. The adversary could gain more votes if a candidate's supporters could be targeted for misprinted ballots.

[2] `http://electionresults.sos.state.mn.us/`

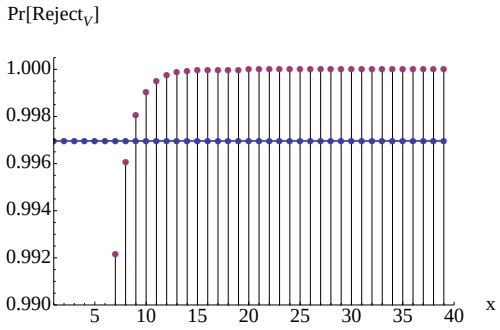


Figure C.1: **The Weakest Link**. The probability that the transcript of an E2E-enabled election will be discovered to be unsound, given ballots were manipulated in an attempt reverse the outcome, is the minimum of two data series: a receipt audit (horizontal line) and tally audit (asymptote). The tally audit converges on 100% as the number of proof instances, x, is increased. The receipt audit however is a function of the number of voters checking their receipts relative to the incidence of fraud. For example, if only a random 5% of voters checked their receipts in a tight race, such as the 2008 Minnesota senate race in which the victory margin was 225 votes, the probability of detecting the fraud would be 99.7%, meaning at $x = 9$, committing to additional instances would not have increased overall effective soundness.

Setting aside misprinted ballots, if say 5% of voters would check their receipts, an incorrect ρ will be detected with 99.7% certainty. Thus x must be increased until the probability of detecting an incorrect τ is greater than this probability, which occurs at $x = 9$ instances. This is shown in Figure C.1. Note that a suitable value of x should be determined prior to the election.[3]

C.1.3 Computational Secrecy

We now prove that Eperio achieves computational secrecy. This relies on the following assumptions:

1. The maximum number of colluding trustees is t.
2. At least one trustee submits to DKG an f_i drawn with uniform randomness from $\mathbb{Z}_q^t$.
3. All outputs are computed with a blackbox.
4. Any polynomial-sized output from $\mathsf{PRNG}(\kappa)$ provides no non-negligible advantage to a PPT-bounded adversary in guessing either κ, the next bit in the output, or any unobserved previous bit.

[3]It is possible to increase the value of x after the election, however this will only increase the probability of detecting cheating in the tally audit, and not in the receipt check or print audit.

5. The function $\mathsf{Commit}(m, r)$ is semantically secure and computationally hiding. That is, given either $c_1 = \mathsf{Commit}(m_1, r_1)$ or $c_2 = \mathsf{Commit}(m_2, r_2)$ for any chosen m_1 and m_2, a PPT-bounded adversary should have no non-negligible advantage in guessing which message was committed to.

Our approach will be to demonstrate that a transcript provides a malicious verifier, $\mathcal{V}'$, with no non-negligible information about which candidate was selected for any ballot in the election. Let $\mathsf{RecoverSel}$ be a PPT-algorithm that returns 1 if it successfully recovers such information. Note that given a correct list of receipts and tally, some information may be determinable. For example, the adversary may be able to determine if a voter abstained or voted randomly (if the adversary instructs voters to, say, always mark the first position). If everyone votes for the same candidate then $\mathsf{RecoverSel}$ will trivially succeed. For reasons such as these, we define a baseline success, ψ, which depends on the security of the voting interface (out-of-scope in our paper) and only theorize that Eperio is computationally secret relative to ψ.

$$\mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho, \tau)) = 1] \triangleq \psi \tag{C.7}$$

Our proof method is a straight-forward adaptation of Okamoto's definition of receipt-freeness [Oka97]. We begin by proving,

$$\begin{aligned} &|\mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho, \tau, \mathbf{C})) = 1] - \\ &\mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho, \tau)) = 1]| \leq \epsilon_1 + \epsilon_2. \end{aligned} \tag{C.8}$$

Assume there exists a polynomial time algorithm $\mathsf{RecoverSel}'()$ that can distinguish these two views with greater than negligible advantage. Such an algorithm could take as input any set of commitments $\mathbf{C}$ and return non-negligible information about the messages committed to. Consider two cases: (i) the random factors in the commitments are indistinguishable from uniform randomness and (ii) the random factors are distinguishable by a bounded $\mathcal{V}'$.

Case (i) is in violation of assumption 5. Given $\mathsf{RecoverSel}'()$ runs in polynomial time, it cannot achieve more information than ϵ. Let this information be ϵ_1. If case (ii) holds, this can be further broken into two cases: (ii-a) the output of the $\mathsf{PRNG}()$ function follows some distinguishable distribution and (ii-b) the seed to the $\mathsf{PRNG}()$ function follows some distinguishable distribution. Case (ii-a) is in violation of assumption 4 which implies that if $\mathsf{RecoverSel}'()$ runs in polynomial time, it cannot achieve more information than ϵ. Let

this information be ϵ_2. Given that $\mathsf{DKG}()$ is information theoretic, case (ii-b) can only hold if all of the trustee shares follow a known distribution, a violation of case (ii-b).

Now consider the following,

$$\begin{aligned}|\mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho,\tau,\mathbf{C},\mathbf{L},\mathbf{E}_{(:,2,:)},\mathbf{R}_z)=1]\\ -\mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho,\tau,\mathbf{C}))=1]|\leq\epsilon_3+\epsilon_4.\end{aligned}\tag{C.9}$$

Again assume $\mathsf{RecoverSel}'()$ can distinguish these two views with greater than ϵ advantage. Such an algorithm could take as input several input-output pairs of a permutation (*e.g.,* from $\mathbf{L}$ or knowledge of one's own receipt) and output non-negligible information about how the permutation maps other pairs not in $\mathbf{L}$. Given that the $\mathsf{Permute}()$ and $\mathsf{PermuteBlock}()$ functions are implemented with perfect shuffles, $\mathsf{RecoverSel}'()$ cannot be polynomial in time unless if the $\mathsf{PRNG}()$, its seed κ, or all the trustee shares follow a distinguishable distribution. As shown by cases (ii-a) and (ii-b) above, this cannot be. Similarly, the information leaked by $\mathbf{R}$ is addressed by case (ii) above.

This demonstrates that the Eperio verification protocol is computationally secret in a single instance. We complete the proof by considering the composition of multiple instances of independently shuffled tables. Give a single instance, k, is secure, consider as an inductive step the addition of $k+1$,

$$\begin{aligned}|\mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho,\tau,\mathbf{C}_{(:,k)},\mathbf{C}_{(:,k+1)},\mathbf{L}_{(:,k)},\\ \mathbf{L}_{(:,k+1)},\mathbf{E}_{(:,2,k)},\mathbf{E}_{(:,2,k+1)},\mathbf{R}_{z,k},\mathbf{R}_{z,k+1})=1]-\\ \mathbf{Pr}[\mathsf{RecoverSel}(\mathtt{View}_{\mathcal{V}'}(\rho,\tau,\mathbf{C}_{(:,k)},\mathbf{L}_{(:,k)},\\ \mathbf{E}_{(:,2,k)},\mathbf{R}_{(z,k)})=1]|\leq\epsilon_5\end{aligned}\tag{C.10}$$

Given the output of instance k, we show how a deterministic polynomial time simulator could generate a second transcript, $k+1$, that is indistinguishable from a real transcript. Intuitively, there is only one set of marks $\mathbf{E}_{(:,2,:)}$ that will satisfy the binary relation $\mathrm{B}(\mathbf{E}_{(i,1,k)},\rho)$. However there are many permutations of $\mathbf{E}_{(i,3,k)}$ that satisfy the binary relation $\mathrm{B}(\mathbf{E}_{(:,2,:)},\tau)$. The output of Eperio only proves that τ is correct, it does not give any information about which receipts the votes for each candidate in τ came from. Any mapping of the results in τ is plausible, and the simulator, $\mathcal{S}_{\mathcal{V}'}$, only needs to generate a transcript of one of them.

$\mathcal{S}_{\mathcal{V}'}$ will generate $\mathbf{E}$ such that $\mathbf{E}_{(i,1,k+1)}$ is randomly shuffled along i for $k+1$. Next, $\mathcal{S}_{\mathcal{V}'}$ will fill in $\mathbf{E}_{(:,2,k+1)}$ according to ρ. Then, $\mathcal{S}_{\mathcal{V}'}$ will fill in $\mathbf{E}_{(i,3,k+1)}$ for each i in $\mathbf{L}_{(i,j)}$

with the appropriate candidate. Finally, $\mathcal{S}_{\mathcal{V}'}$ will randomly assign candidates to the rest of $\mathbf{E}_{(i,3,k+1)}$ such that τ is satisfied. The simulator will commit to this, generating $\mathbf{C}_{(i,j)}$.

The verifier, $\mathcal{V}'$, will observe random bits $\mathbf{Z}_i$ and challenge $\mathcal{S}_{\mathcal{V}'}$ to reveal $\mathbf{R}_{(z+1,j)}$. Since we have gone through the difficulty of validating all of the public parameters, $\mathcal{S}_{\mathcal{V}'}$'s output will be acceptable at this stage with probability 1. It does not need to rewind the protocol.

Consider a bound on the size of $\mathcal{E}$ for a given Ω. Recall that b is the number of ballots, s the number of selections, and x the number of instances. The construction of $\mathbf{E}_{(i,j,k)}$ requires two permutations, $\Pi_S = s!^b$ and $\Pi_E = (bs)!^x$, while $\mathbf{R}_{(i,j)}$ requires two l-bit random factors for each instance, yielding 2^{2xl} possible values. The number of random bits for any $\mathcal{E}$ is thus on the order of $2xl + b \cdot \mathcal{O}(s \log s) + x \cdot \mathcal{O}(bs \log bs)$. For large enough b and s, this is much greater than the l bits of true randomness in κ. Thus, only a small subset of $\mathcal{E}$ are possible outputs of the described protocol. Due to the properties of the PRNG, a bounded simulation strategy cannot distinguish this subset from a randomly generated $\mathcal{E}$ however a unbounded $\mathcal{V}'$ can. This is the intuition behind Eperio offering only *computational* secrecy.

C.1.4 Everlasting Privacy

We now briefly sketch how to modify Eperio to achieve everlasting privacy. It requires three simple modifications. First, the commitment function is replaced with one that is perfectly hiding and computationally binding. This affects the soundness proof as committed values can be opened, by an unbounded adversary, to any value. Therefore the soundness of the argument is only as hard as the computational bindingness of the commitments. This also changes assumption 5 of the secrecy proof. Second, trustees generate all of the randomness needed to do the permutations and random factors. From a practical standpoint, this significantly increases the complexity of the protocol as a DKG is needed to threshold share every piece of randomness. Finally, given the direct source of randomness, the PRNG can be eliminated. This removes assumption 4 of the secrecy proof.

The first 3 secrecy assumptions still hold, demonstrating that everlasting privacy is achievable but only within a narrow view of the protocol. For example, if at least $t + 1$ trustees reveal their private inputs, privacy can still be broken. This holds for any distributed voting system in the literature.

C.2 Proof of Theorem 3

Proof of Theorem 3. To address the hidingness, consider an attack game where an adversary A provides equal-length messages M_1 and M_2 to an oracle O. O generates a

random K and IV, flips a coin z, and returns $C_z = E(K, \text{IV}, M_z)$. A must guess if C_z is a commitment to M_1 or M_2 (or that both are possible under a different K). We say Commit is *computationally hiding* if the adversary's advantage over a random guess is negligible in k. Assume an algorithm $z \leftarrow \mathcal{A}_h(m_1, m_2, c_z, \text{IV})$ exists that can guess z correctly with non-negligible advantage. It is straightforward to see how this algorithm would provide an advantage in a chosen plaintext attack game: the adversary would submit the CPA challenge values to $\mathcal{A}_h$ and respond with z. Since E is CPA-secure under Theorem 1, $\mathcal{A}_h$ can be assumed to be not computable by a PPT-bounded adversary. Thus the commitment scheme is computationally hiding.

We say Commit is *statistically binding* for a correctly formed M' if,

$$p \triangleq \mathbf{Pr}[\exists M' \text{ s.t. } (C, \text{IV}) = \mathsf{Commit}(M', K') | (C, \text{IV}) = \mathsf{Commit}(M, K) \ \&\ M \neq M'] \leq \mathsf{negl}(k). \tag{C.11}$$

Recall redundancy $f(M) = M \| M$; that is, $\mathsf{Commit}(M, K) = E(K, \text{IV}, M \| M \| M)$.[4] If Conjecture 2 is true, the probability p that M' and K', as defined in Equation C.11, exist given some distinct M and K can be determined. A random function has the following form, where x is the size of domain of the commitment function and y, the co-domain.[5]

$$p = 1 - (1 - \frac{1}{y})^{x-1} \tag{C.12}$$

Recall the input to $\mathcal{E}$ is a m-bit message block and a k-bit key. For a message M with L blocks ($|M| = L \cdot m$), the domain of Commit is $x = 2^{k+Lm}$. The 2 repetitions of M are fixed by choice of the domain, and the initialization vector is a constant. The co-domain is $y = 2^{3Lm}$, with the factor of 3 due to the redundancy function f. We are interested in an upper-bound to equation C.12 with these values of x and y. As we are not trying to prove a general case, recall our assumption that $m = k$, and note that L is a positive integer such that p is maximized when L is minimized. The upper bound at $L = 1$ is given as Equation C.13.

[4] We would like to thank and acknowledge Ronald L. Rivest for suggesting this approach in creating a binding commitment from a block cipher. Any errors in the analysis are our own.

[5] *Intuition:* It may help to conceptualize this as having x balls and y bins. We throw the first ball into a random bin, and ask for the probability that at least one of the remaining balls will be thrown into the same bin.

$$
\begin{aligned}
p &\leq 1 - (1 - \frac{1}{2^{3k}})^{2^{2k}-1} && \text{(C.13)} \\
&\approx 1 - \exp(-\frac{2^{2k}}{2^{3k}}) && \text{(C.14)} \\
&= 1 - \exp(-2^{-k}) && \text{(C.15)} \\
&\leq \mathsf{negl}(k) && \text{(C.16)}
\end{aligned}
$$

Equation C.14 follows from the approximations that $(1 - 1/y)^x \approx \exp(-x/y)$ and $(1 - 1/y)^{-1} \approx 1$ for large y. □

We have demonstrated a very specific statistically hiding commitment function that can be constructed from a block cipher, assuming conjecture 2 holds.

Appendix D

Security Proof and Notes for Selections

D.1 Registration

The Registration protocol is a cut-and-choose argument given by registrar $\mathcal{R}$ to $\mathcal{V}_i$ for $\{(c, r) : c' = \mathsf{ReRand}_e(c, r)\}$. It is a variant of Benaloh's voter initiated auditing [Ben06]. The protocol actually includes $\alpha > 1$ ciphertexts, $c_1, \ldots, c_\alpha$, and the s^{th} one will be chosen for submission. In discussing general properties of Registration, we refer to c_s as simply c to keep the discussion applicable to protocols that do not have a set of ciphertexts (*i.e.,* are not based on cut-and-choose).

We first sketch informally several properties one may want from a registration (and voting) protocol.

1. *Integrity*: $\mathcal{V}_i$ should be convinced that c' rerandomizes c.
2. *Secrecy*: An adversary $\mathcal{A}$ should not be able to determine that c was rerandomized from the output of the protocol.
3. *Receipt-Free*: $\mathcal{A}$ in collusion with $\mathcal{V}_i$ should not be able to determine that c was rerandomized from any of the following: supplying inputs, examining the output, or examining a transcript of the protocol kept by $\mathcal{V}_i$.
4. *Covert-Free*: $\mathcal{R}$ cannot inject information into the transcript that would be useful for coercion or breaking secrecy.
5. *Dispute-Free*: If $\mathcal{R}$ does not follow the protocol, $\mathcal{V}_i$ can prove the protocol was not followed.
6. *Bare-Handed*: $\mathcal{V}_i$ can complete the protocol without performing computations during the protocol.

We will show that the protocol we provide has integrity (correctness and soundness), secrecy, receipt-freeness and is bare-handed. We do not attempt to prevent covert chan-

nels in the basic protocol but provide some discussion toward this point, as well as only discussing disputes.

Given a transcript of the Registration protocol, the c' can be either accepted or rejected as a rerandomization of c. If Registration is run correctly, the decision will always be to accept (completeness). If the protocol is not correct, the decision will be to reject with a high probability (soundness). Finally, the outputs do not provide any information that can be used by a computationally bounded adversary to determine any non-negligible information about the secrets in the protocol: c and r (computational secrecy). Finally, we show $\mathcal{V}_i$ has a coercion-resistant strategy (receipt-freeness).

Assumptions. We make the following assumptions regarding the security of Registration:

1. An adversary $\mathcal{A}$ can corrupt $\mathcal{V}_i$ prior to the protocol and provide $\mathcal{V}_i$ with inputs to the protocol,
2. $\mathcal{A}$ can corrupt $\mathcal{V}_i$ prior to the protocol and require $\mathcal{V}_i$ to provide a plausible transcript of the protocol afterward,
3. The protocol between $\mathcal{V}_i$ and $\mathcal{R}$ is run over an untappable channel unaccessible to $\mathcal{A}$,
4. V_i complies with erasures,
5. $\mathcal{A}$ cannot corrupt $\mathcal{R}$, and
6. $\mathcal{A}$ is bounded to probabilistic polynomial time (PPT).

For the proof, $\mathcal{R}$ is unbounded in computation power, while $\mathcal{V}_i$ is PPT-bounded. These bounds are used to demonstrate that soundness is statistical and not computational. Either entity may employ a malicious strategy and we denote this with a prime $(\mathcal{R}', \mathcal{V}_i')$.

Completeness. If $\mathcal{R}$ follows the protocol, he will produce correct rerandomizations for each of the α ciphertexts, including the one chosen. $\mathcal{V}_i$ will accept this transcript by checking the $\alpha - 1$ ciphertexts and finding they are correct. Note that we are demonstrating the protocol is complete, not the rerandomization of the chosen ciphertext. It is possible that the rerandomization for the chosen ciphertext is correct but it is not correct for another ciphertext. In this case, the $\mathcal{V}_i$ will reject the transcript despite it being correct for the only value of importance. Completeness does not capture false-negatives.

Soundness. We now consider whether $\mathcal{V}_i$ will ever accept a transcript when c' is not a rerandomization of c. If the probability of rejecting a false proof is high (but not necessarily overwhelming), we say Registration is sound. Let c_s be the ciphertext chosen for submission and let c_{-s} be one ciphertext that is not chosen for submission.

We assume that the transcript will be checked by $\mathcal{V}_i$. In reality, only a fraction of $\mathcal{V}_i$'s will check. Any probabilities should be appropriately scaled according to the fraction of $\mathcal{V}_i$'s checking. For simplicity, we omit this factor.

A malicious $\mathcal{R}'$ rerandomizes $c_1, \ldots, c_\alpha$ before learning the value of s. If $\mathcal{R}'$ incorrectly rerandomizes more than one ciphertext, the transcript will be rejected with certainty. Therefore $\mathcal{R}'$ must choose one. If $\mathcal{V}_i$ chooses s at random, $\mathcal{R}'$ can do no better than choosing randomly. Denote $\mathcal{R}'$'s choice as $c_{\hat{s}}$. If $s \neq \hat{s}$, $\mathcal{V}_i$ rejects $\mathcal{R}'$'s false transcript. If $s = \hat{s}$, $\mathcal{V}_i$ accepts $\mathcal{R}'$'s false transcript. The probability $\mathbf{Pr}[s = \hat{s}] = \alpha^{-1}$. Therefore, the soundness of Registrar is:

$$\mathbf{Pr}[\texttt{REJECT}_{(\mathcal{R}',\mathcal{V}_i)}] \;=\; 1 - \alpha^{-1}. \tag{D.1}$$

If $\mathcal{R}'$ creates false transcripts for k voters, the probability of at least one $\mathcal{V}_1, \ldots, \mathcal{V}_k$ rejecting the false transcript is:

$$\mathbf{Pr}[\texttt{REJECT}_{(\mathcal{R}',\mathcal{V}_1)} \vee \ldots \vee \texttt{REJECT}_{(\mathcal{R}',\mathcal{V}_k)}] \;=\; 1 - \alpha^{-k}. \tag{D.2}$$

For example, to achieve a probability of detection of at least 99.9%, $\{\alpha = 10, k = 3\}$ or $\{\alpha = 2, k = 10\}$ are sufficient.

Computational Secrecy. Under assumption 4, $\mathcal{V}_i$ complies with erasures. We assume additionally under this compliance that $\mathcal{V}_i$ cannot commit the values to memory. We do not consider here the issue of a voter memorizing a few bits of the values, which could make for interesting future work. We also note that while the voter could record the values with a device, this device could also be used to record how they vote in an in-person voting protocol. We only claim our system to be as secure as the baseline of an in-person voting protocol. Erasures can be enforced by having an erasure confirmation code printed under the scratch-off. In Section D.4, we collect features for enhanced verification cards to be explored in future work.

With erasures, a transcript of the protocol does not contain the values (c_s, r_s) for c'_s. However $\mathcal{V}'_i$ (*i.e.,* malicious $\mathcal{V}_i$) has the value c_s. Therefore $\mathcal{V}'_i$ can assert that the value of c_s is $\hat{c}_s$, which may or may not be true. The adversary must decide if the asserted $\hat{c}_s$ and the rerandomized value c'_s encrypt the same plaintext ρ. Toward a contradiction, assume the adversary has an efficient plaintext equality (peq) algorithm $\{\mathtt{T}, \mathtt{F}\} \leftarrow A^{\text{PEQ}}(\hat{c}_s, c'_s)$ to determine this equality.

$\mathcal{A}$ can use A^{PEQ} to win the CPA game. In the CPA game, $\mathcal{A}$ provides p_0 and p_1 to oracle $\mathcal{O}$, $\mathcal{O}$ flips coin $b \leftarrow_r \{0, 1\}$, and returns challenge ciphertext $c_b = \mathsf{Enc}_e(p_b, r)$. $\mathcal{A}$ encrypts

p_0 as c_0 and submits queries $A^{\text{PEQ}}(c_0, c_b)$. If T, $\mathcal{A}$ guesses $b = 0$ and otherwise $b = 1$. Since the encryption is CPA-secure, $A^{\text{PEQ}}(\hat{c}_s, c'_s)$ cannot be efficient against the encryption and the PPT-bounded adversary (assumption 6) cannot decide if $\mathcal{V}'_i$ asserted a correct value.

Receipt-Freeness. Under assumption 1, $\mathcal{A}$ can provide inputs for $\mathcal{V}'_i$. For example, $\mathcal{A}$ could provide a list of α ciphertexts $c_1, \ldots, c_\alpha$ for $\mathcal{V}'_i$ to use. $\mathcal{V}'_i$ however can replace one of $\mathcal{A}$'s ciphertexts with her desired c_s and choose it for submission. After the erasure, $\mathcal{A}$ cannot distinguish if she supplied his ciphertext or her own ciphertext under the argument given above for computational secrecy.

However, $\mathcal{V}'_i$ has one further input to the protocol and that is the value of s itself. $\mathcal{A}$ can exploit this with a coercion contract. Assume $\mathcal{A}$ provides $\mathcal{V}'_i$ with α ciphertexts and $\mathcal{V}'_i$ substitutes her own for one of them. Further assume that a communication channel exists between $\mathcal{A}$ and $\mathcal{V}'_i$. If such a channel existed, $\mathcal{V}'_i$ could tell $\mathcal{A}$ the values she received and $\mathcal{A}$ could tell her a value of s. Unless if $\mathcal{V}_i$ placed her substituted value in the correct place, her non-compliance will be caught.

$\mathcal{A}$'s best strategy is to chose a random value for s. The goal of a coercion contract is to replace the interactive channel between $\mathcal{A}$ and $\mathcal{V}'_i$ (which does not actually exist) with a non-interactive call to a random oracle made by the voter to receive s and re-queried by $\mathcal{A}$ to verify compliance. For example, $\mathcal{A}$ could require $\mathcal{V}_i$ to submit the smallest rerandomized value c'_i generated by $\mathcal{R}$. If this generation is random, then the implied value of s will be random as well. Since the voter has to choose where to substitute her value before seeing what such an s will be, she will get caught with probability $1 - \alpha^{-1}$.

In order to provide receipt-freeness, Registration allows $\mathcal{V}_i$ to rewind the protocol upon seeing the list of rerandomized values. In practice, they could be displayed and rewound until satisfactory and only then printed. An alternative approach is to have the voter commit to s prior to the protocol. This could be accomplished bare-handed by having the voter mark beside the cell she will scratch-off or putting s in a sealed envelope, which will be checked when she leaves.

Covert-Freeness. The protocol does not protect against covert channels. The issue of covert channels has been addressed in the literature with verifiable random functions [GGR09] or pre-committed randomness [FB09]. However if $\mathcal{R}$ is printing values, it is very easy for $\mathcal{R}$ to leak information with slight variations in how things are printed. Further, $\mathcal{R}$ cannot prove anything that is interesting to the adversary. It can assert what $\mathcal{V}'_i$ supplied it with but these values could be easily simulated: the erased values are c and r' for c'. A simulator can generate such values: just rerandomize c' with r'' to get c'' and claim c'' and r'' were submitted by the voter.

Since under assumption 5 $\mathcal{A}$ cannot corrupt $\mathcal{R}$, $\mathcal{A}$ cannot trust any assertions leaked covertly by $\mathcal{R}$ on the face of the assertion itself. The assertions must include a sound argument for (or proof of) what is being asserted.

D.2 Coercion-Resistance

We now turn our attention to the entire protocol and consider whether it is coercion-resistant under the game-based definition from Juels *et al.* [JCJ05]. Moving forward, we will model registration as an abstract protocol possessing the properties demonstrated. Following Juels *et al.*, we will also replace portions of the protocol with idealized primitives. These include the password-based key derivation function, mixing, plaintext equality tests, zero-knowledge proofs of knowledge, and hash functions used to make the proofs simultaneous, non-malleable, and non-interactive (*i.e.,* secure against a dishonest verifier).

Other cryptographic definitions of coercion-resistance (or the related receipt-freeness) exist in the literature. Kusters *et al.* provide another game-based definition [KTV10b] that can be seen as a generalization of the Juels *et al.* definition to voting systems that do not fit the general architecture of the Juels *et al.* scheme (Selections does fit this general architecture). Gardner *et al.* provide a definition focused more on the elimination of covert channels and it is specific to the architecture of Benaloh's voter initiated auditing [GGR09]. Other researchers have used simulation-based proofs [MN07, TRN08, UM10]. See Kusters *et al.* [KTV10b] for an excellent overview of the different approaches as of 2010.

D.2.1 Security Game

High-level Overview. We present the experiment $\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}$ in Algorithm 11 and experiment $\mathbf{Exp}^{\text{cr-ideal}}_{ES,\mathcal{A}'}$ in Algorithm 12. These experiments are quite faithful to the original experiments proposed by Juels *et al.*, except we have modified them to correspond to user-chosen passwords instead of registrar-chosen credentials. We have replaced some of the notation to better integrate with Selections as presented in Section 11.4. Here we will briefly describe the game.

The game involves a challenger running the game, an adversary $\mathcal{A}$, and a set of voters. There are n_V voters and they are partitioned into three sets throughout the game. At the beginning of the protocol, the adversary corrupts n_A voters. We call this set of corrupted voters V and the remaining uncorrupted voters are U. The adversary then specifies a single voter from U to coerce. We call this voter j. So the makeup of the voters is n_A corrupted voters, 1 coerced voter, and $n_V - n_A - 1$ uncoerced (or honest) voters.

Algorithm 11: Experiment $\mathbf{Exp}^{\mathrm{cr}}_{ES,\mathcal{A}}$

Input: Security parameters k_1 (encryption) and k_2 (mixing), number of voters n_V, number of corrupted voters n_A, trustees public key e, distribution of types of votes cast by honest voters $\mathcal{D}$.

Private Input (T_i): Private key shares constituting $d_{\mathcal{T}}$.

$(V, U) \leftarrow \mathcal{A}(\texttt{VoterIDs}, \text{“corrupt”})$ //$\mathcal{A}$ corrupts n_A voters

$\{(\rho_i, \mathsf{Enc}_e(g^{\rho_i})) \leftarrow \mathsf{Register}(i, e, k_1)\}_{i=1}^{n_V}$ //All voters register

$\{\rho_i\}_{i \in V} \leftarrow \mathcal{A}(\texttt{VoterIDs}, \text{“obtain”})$ //$\mathcal{A}$ obtains corrupted passwords

$j \leftarrow \mathcal{A}(\texttt{VoterID}, \text{“coerce”})$ //$\mathcal{A}$ coerces one voter j

Check $|V| = n_A$ and $j \notin V$ (upon failure, output $\perp$) //Validity check on $\mathcal{A}$

$b \leftarrow_r \{0, 1\}$ //Flip coin b

if $b = 0$ **then**

  $\mathsf{AllVotes} \Leftarrow \mathsf{Cast}(\rho_j, \mathbf{B}_j, \beta_j, k_1)$ //Voter casts ballot $\mathbf{B}$

  $\rho^* \leftarrow \mathsf{PanicPassword}(\rho_j)$ //Voter sets ρ^* to a panic password

else

  $\rho^* \leftarrow \rho_j$ //Voter sets ρ^* to real registered password

$\mathsf{AllVotes} \Leftarrow \{\mathsf{Cast}(\rho_i, \mathbf{B}_i, \beta_i, k_1, \mathcal{D})\}_{i \neq j, i \notin V}$ //Honest voters cast ballots

$\mathsf{AllVotes} \Leftarrow \mathcal{A}(\{\rho_i, \mathbf{B}_i, \beta_i, k_1\}_{i \neq j, i \in V}, \text{“cast”})$ //$\mathcal{A}$ casts corrupt ballots

$\mathsf{AllVotes} \Leftarrow \mathcal{A}(\rho^*, \mathbf{B}, \beta, k_1, \text{“cast”})$ //$\mathcal{A}$ casts coerced ballot

$(\mathsf{ValidVotes}, \Pi, \Gamma) \leftarrow \mathsf{PreTally}(\mathsf{Roster}, \mathsf{AllVotes}, d_{\mathcal{T}}, k_2)$ //PreTally is conducted

$b' = \mathcal{A}(\mathsf{ValidVotes}, \Pi, \Gamma, \text{“guess”})$ //$\mathcal{A}$ guesses

Output 1 if $b' = b$, else 0.

The adversary is non-adaptive: he must specify at the beginning of the protocol which voters are to be corrupted. He does this with a “corrupt” command. All voters register. The adversary is given the passwords registered by the corrupt voters (using the “obtain” command). The adversary then chooses the voter to coerce (using the “coerce” command). This voter cannot already be corrupted (*i.e.,* the adversary does not know her registered password). The challenger then makes sure the adversary did not cheat in how voters it corrupted or who it selected to coerce.

At the heart of the game, the adversary will decide between one of two scenarios selected with uniform randomness by the challenger. In the first scenario, the coerced voter does everything she can to escape coercion: she posts the ballot she wants to using her real password and provides a panic password to the adversary. In the second scenario, the coerced voter does everything she can to cooperate with the adversary: she does not cast

Algorithm 12: Experiment $\mathbf{Exp}^{\text{cr-ideal}}_{ES,\mathcal{A}'}$

$(V, U) \leftarrow \mathcal{A}'(\texttt{VoterIDs}, \text{“corrupt”})$
$\{(\rho_i, \mathsf{Enc}_e(g^{\rho_i})) \leftarrow \mathsf{Register}(i, e, k_1)\}_{i=1}^{n_V}$
$\square$ //$\mathcal{A}'$ does not obtain corrupted passwords
$j \leftarrow \mathcal{A}'(\texttt{VoterID}, \text{“coerce”})$
Check $|V| = n_A$ and $j \notin V$ (upon failure, output $\perp$)

$b \leftarrow_r \{0, 1\}$
if $b = 0$ **then**
  $\mathsf{AllVotes} \Leftarrow \mathsf{Cast}(\rho_j, \mathbf{B}_j, \beta_j, k_1)$
  $\boxed{\rho^* \leftarrow \rho_j}$ //$\mathcal{A}'$ always gets real password
else
  $\rho^* \leftarrow \rho_j$ //$\mathcal{A}'$ always gets real password

$\mathsf{AllVotes} \Leftarrow \{\mathsf{Cast}(\rho_i, \mathbf{B}_i, \beta_i, k_1, \mathcal{D})\}_{i \neq j, i \notin V}$
$\mathsf{AllVotes} \Leftarrow \mathcal{A}'(\{\square, \mathbf{B}_i, \square, k_1\}_{i \neq j, i \in V}, \text{“cast”})$ //$\mathcal{A}'$ just submits intent
$\mathsf{AllVotes} \Leftarrow \mathcal{A}'(\rho^*, \mathbf{B}, \square, k_1, \text{“cast”})$ //$\mathcal{A}'$ does not see AllVotes
$(\mathsf{ValidVotes}, \Pi, \Gamma) \leftarrow \boxed{\mathsf{IdealPreTally}}(\mathsf{Roster}, \mathsf{AllVotes}, d_{\mathcal{T}}, k_2)$ //Idealized

$b' = \mathcal{A}'(\mathsf{ValidVotes}, \square, \Gamma, \text{“guess”})$ //$\mathcal{A}'$ guesses without Π
Output 1 if $b' = b$, else 0.

Remarks: Boxed items represent changes from $\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}$.

any votes and she provides the adversary with her real password.

The election then proceeds with the honest voters casting their ballots, the adversary casting ballots on behalf of the corrupted voters (while having a full view of what the honest voters cast), and finally the adversary casting a ballot on behalf of the coerced voter using the password supplied to him (both through the “cast” command). The votes go through the pre-tally, and list of encrypted valid votes is returned, along with Π, a full transcript of the entire PreTally process, including all the proofs, and Γ which replicates a subset of the information in Π (see below). Based on this information, the adversary guesses which scenario. If the adversary cannot distinguish the scenarios, then he effectively cannot distinguish between the success or failure of his coercion.

Ideal Game. We can ask ourselves, what is the highest level of security we can achieve? It is tempting to conclude it is the scenario where a bounded adversary cannot distinguish between the scenarios. In other words, his advantage at guessing is ϵ-close to $1/2$. Juels

et al. point out this not always the case. In systems like JCJ, Civitas, and AFT, the final output of the protocol is the tally. In Selections, the protocol ends before this final decryption stage however consider for a moment that it produced the final tally as well. If the adversary were to cast a ballot for Alice using the password provided to him by the coerced voter and the final tally produced no votes for Alice, then the adversary could distinguish with certainty which scenario occurred in $\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}$.

However from this, we should not conclude anything about the coercion-resistance of Selections: the distinguishability of the scenarios in this case is invariant to the election system used. Since we are interested in the advantage an adversary would have with a particular election system, we need to measure it against an experiment that preserves the distinguishers that are invariant to the election system. To this end, Juels *et al.* define $\mathbf{Exp}^{\text{cr}-\text{ideal}}_{ES,\mathcal{A}'}$ where the election is conducted in a very similar manner but the adversary does not get any access to the cryptographic data involved.

In Selections, because a final tally is not produced, this illustrative distinguisher does not work. However, the adversary does learn Γ which contains four quantities: the number of cast votes, number of submissions eliminated due to invalid proofs, the number of submissions eliminated because they were duplicates, and the number of submissions eliminated for having invalid passwords. The last quantity could be distinguishing: if no other honest voters cast ballots in the election and the adversary abstains from casting the coerced voter's vote, he will learn if the coerced voter cast a vote of her own by the number of valid votes (relative to the number he cast through corrupted voters). In general, the number of votes cast will always be greater by one when the coin flip is 0, as opposed to 1. In order to negate the significance of this fact on the adversary's advantage, we introduce a notion of adversarial uncertainty.

Adversarial Uncertainty. If the adversary knows how all the other voters will vote, then he can easily and consistently win the game even without seeing the final tally. He will simply count how many votes were submitted. This number differs according to the coin flip. Juels *et al.* address this by specifying a distribution $\mathcal{D}$ over the honest voter's votes which specifies the number of ballots they will cast (voters can abstain, submit one, or submit more than one), and whether submitted ballots contain a valid proof and/or use a valid credential. $\mathcal{D}$ acts a "noise" that masks the coerced voter's actions. Additionally, they note, noise can be injected into the system by any interested party.

In Selections, voters rerandomize their registered encrypted password from Roster and include it in their vote, but do not reveal it is from their entry in Roster. They do this by forming an anonymity set of $\beta - 1$ other entries. When the size of β is not specified by the system (Case 3 below), we assume that $\mathcal{D}$ also specifies a distribution for the β values used by voters. It is likely to be bimodal, with voters either using the minimum β (for

efficiency) or the largest β (for maximal privacy). We assume adversarial uncertainty with respect to this distribution.

D.2.2 Ideal Components

- $\widetilde{DKG}(k_1)$ is a distributed key generation algorithm that shares a secret decryption key. In the idealized form, we assume only the challenger learns the key. k_1 is a security parameter.
- $\widetilde{PBKDF}(\hat{\rho})$ is a password-based key derivation function. It takes input from an unspecified distribution and returns a value that is computationally indistinguishable from a uniform random selection in $\mathbb{G}_q$. It may rely on a random oracle.
- $\widetilde{POK}1(\beta)$ is a proof of knowledge that a given ciphertext reencrypts 1-out-of-β other ciphertexts. The computational secrecy of the witness is ϵ_{wh-pok} ("wh" for witness hiding) and it is non-interactive through a random oracle assumption. Using standard techniques, the component can output a simulated proof (by control of the random oracle) or the witness can be extracted (through rewinds).
- $\widetilde{POK}2$ is a proof of knowledge of a discrete log. It has the same properties as $\widetilde{POK}1$.
- $\widetilde{MIX}(\mathsf{UniqueVotes}, k_2)$ is a mix network. In the idealized form, it shuffles and rerandomizes a list of ciphertexts. k_2 is a security parameter that describes how unlinkable an output ciphertext is from its corresponding input (it differs from k_1 because some implementations reveal partial information about the permutation used).
- $\widetilde{PET}$ is a plaintext equality test. In the idealized form, it simply returns a bit describing whether two ciphertexts encrypt the same message or not.

D.2.3 Case 1: $\beta = R$

We define the advantage of $\mathcal{A}$, where an output of 1 is the adversary correctly stating b, as,

$$\mathbf{adv}^{\mathrm{cr}}_{ES,\mathcal{A}} = |\mathbf{Pr}[\mathbf{Exp}^{\mathrm{cr}}_{ES,\mathcal{A}}(\cdot) = 1] - \mathbf{Pr}[\mathbf{Exp}^{\mathrm{cr-ideal}}_{ES,\mathcal{A}'}(\cdot) = 1]|.$$

Recall that when voter submit a ballot, they prove they re-encrypted one of β registered passwords. β represents the size of the anonymity set for the voter. We show that when β is the full roster R for all voters, $\mathbf{adv}^{\mathrm{cr}}_{ES,\mathcal{A}}$ for Selections is negligible. Setting $\beta = R$ does

impact performance. Vote casting is linear in the size of the ElectionRoster and Pre-Tallying is quadratic.

We approach the proof through a series of modified games, beginning with a detailed implementation of $\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}$ in Game 0 and ending with a game that, from inspection, has the same advantage as $\mathbf{Exp}^{\text{cr-ideal}}_{ES,\mathcal{A}'}$. For each modification, we will show that $\mathcal{A}$'s advantage in distinguishing which game he is playing is bounded by a quantity that is negligible in some security parameter.

Game 0. We now describe the initial game. Note that the item numbers in this game correspond to the same step in $\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}$ and $\mathbf{Exp}^{\text{cr-ideal}}_{ES,\mathcal{A}'}$. For modified games, we will only include the modified lines.

Set-up assumptions. The experiment takes as input a number of parameters that were generated during some set-up phase. This includes a uniformly random private key $d_{\mathcal{T}} \leftarrow_r \mathbb{Z}_q$ which we assume is output in shares to a set of trustees with the $\widetilde{DKG}(k_1)$ oracle. The corresponding public key $e \in \mathbb{G}_q$ is published. Security parameter k_1 is specified such that computing d_τ from e is $\mathsf{negl}(k_1)$. A generator of $g_0 \leftarrow_r \mathbb{G}_q$ is also chosen by the trustees at random.

1. **Corrupt voters.** The adversary $\mathcal{A}$ selections a subset of the n_V eligible voters to corrupt. He outputs two sets: U the uncorrupted voters and V the corrupted voters.

2. **Registration**. For each voter i, the challenger chooses a random password $\hat{\rho}$, queries $\widetilde{PBKDF}(\hat{\rho})$ and obtains a random ρ. With $r \leftarrow_r \mathbb{Z}_q$, the challenger encrypts ρ under key e. The output is Roster, which includes for each voter an entry of the form: $\langle \mathtt{VoterID}_i, \mathsf{Enc}_e(g_0^{\rho_i}, r_i)\rangle$.

3. **Obtain corrupted passwords**. For each voter in V, the challenger supplies the adversary with the passwords: $\langle \rho_i \rangle_{i \in V}$ and a proof of correctness using $\widetilde{POK1}(1)$.

4. **Choose coercion target**. From the uncorrupted voters U, $\mathcal{A}$ selects voter j to coerce. Voter j is removed from set U.

5. **Validity Check**. The challenger checks that the subset of corrupted voters is according to input parameter n_A and that $\mathcal{A}$ did not obtain the password for the coerced voter j in step 4. If any test fails, the game halts and outputs $\perp$.

6. **Flip coin**. The challenger flips a coin $b \leftarrow_r \{0, 1\}$.

7. **Heads: Cast real ballot**. If $b = 0$, the challenger constructs a ballot for coerced voter j using password ρ_j. Let $c_j = \mathsf{Enc}_e(g_0^{\rho_j}, r_j)$ for voter j from Roster and c'_j be

a rerandomization of c_j. The challenger appends to $\mathsf{AllVotes}$ a ballot of the form $\langle g_0^{\rho_j}, \mathsf{ReRand}(c_j), \mathbf{B}_j, \pi_1, \pi_2 \rangle$ where π_1 is the output of $\widetilde{POK1}(\beta)$ and π_2 is the output of $\widetilde{POK2}$.[1] Recall our assumption that the knowledge error for both oracles is $\mathsf{negl}(k_1)$, the same security parameter for the encryption. Also note that due to the use of an append operation, the adversary knows exactly where this ballot, if it exists, will be posted—a point we will return to later.

8. **Heads: Generate panic password**. Continuing the case that $b = 0$, the challenger chooses a random password $\hat{\rho}^*$, queries $\widetilde{PBKDF}(\hat{\rho}^*)$, and obtains a random ρ^*. He sets $\rho_{(\mathcal{A},j)} = \rho^*$ and provides the adversary with $\rho_{(\mathcal{A},j)}$.

9. **Tails: Give real password**. The challenger sets $\rho_{(\mathcal{A},j)} = \rho_j$ and provides the adversary with $\rho_{(\mathcal{A},j)}$.

10. **Honest voters vote**. For each voter remaining in U, the challenger posts a ballot following the procedure used in step 7 for the coerced voter. The output to $\mathsf{AllVotes}$ is $\langle g_0^{\rho_i}, \mathsf{ReRand}(c_i), \mathbf{B}_i, \pi_1, \pi_2 \rangle_{i \in U}$.

11. **Corrupt voters vote**. $\mathcal{A}$ constructs votes for the remaining candidates. The adversary is not bound to the protocol and can post tuples of any form (*e.g.,* duplicate votes, invalid votes, votes with invalid proofs, *etc.*). These are provided to the challenger, who appends them to $\mathsf{AllVotes}$.

12. **Coerced voter votes**. $\mathcal{A}$ constructs one remain vote for the coerced voter j, potentially using the $\rho_{(\mathcal{A},j)}$ he was provided with.

13. **PreTally**. The challenger operates the $\mathsf{PreTally}$ protocol. It is detailed earlier in the paper, so we only briefly outline it here. The challenger first verifies the proofs and removes tuples with invalid proofs, producing $\mathsf{ProvedVotes}$. The challenger then inspects the tuples for duplicate values for g^{ρ_i}. For pairs with the same value, he only retains the most recent. The new list is $\mathsf{UniqueVotes}$. He queries $\widetilde{MIX}(\mathsf{UniqueVotes}, k_2)$ which produces a shuffled output and proof of correctness. Each output entry is distinguishable from its corresponding input by a $\mathsf{negl}(k_2)$ factor. Finally, for each tuple, the first two entries are supplied to $\widetilde{PET}$. If the test is negative, the entry is removed. The remaining list $\mathsf{ValidVotes}$ is output to $\mathcal{A}$ along with each preceding list.

14. **Guess**. The adversary produces a guess $b' \in \{0, 1\}$. If $b' = b$, Game 0 outputs a 1. Otherwise, it outputs 0.

[1] Note that in JCJ, the adversary can specify the candidate the voter votes (so that the security does not depend on the adversary not knowing) for however in Selections, we never decrypt the ballots and we leave ballot information $\mathbf{B}$ generic. However, we if assume $\mathbf{B}$ to be a ciphertext, we can allow the adversary to choose the corresponding plaintext.

Intuition. The intuition behind our modifications is as follows. The coerced voter behaves differently in at least two regards depending on the coin flip: (1) she submits a second vote and (2) she supplies either a real or panic password to the adversary. The second vote has two potentially distinguishing features: (1a) the rerandomized value from the Roster belongs to the voter[2] and (1b) the asserted password matches the value on the Roster. Game 1 addresses (1a) and (1b) and Game 2 addresses (2). These games will show that a bounded adversary cannot use values associated with the coerced voter directly. However he could potentially learn the actions of all the honest voters in the system. If he were successful, he would know the actions of everyone except the coerced voter and could then indirectly determine the actions of the coerced voter based on the result. We address the honest voters in Game 3.

Game 1. For Game i, define $S_i = \mathbf{Pr}[\text{Game i} = 1]$. We describe Game 1 and show that $|S_1 - S_0| \leq \epsilon_{cpa} + \epsilon_{wh-pok} + \epsilon_{ddh} + \epsilon_{wh-pok}$.

2 **Registration**. Registration proceeds as in Game 0.

6 **Flip coin**. Coin is flipped as in Game 0.

7 **Heads: Cast ballot**. The challenger chooses a random value $\hat{z} \in \mathbb{Z}_q$, computes $\hat{c} = \mathsf{Enc}_e(g_0^{\hat{z}}, r)$, and simulates a proof $\hat{\pi}_1$ that $\hat{c}$ is one of β entries on the Roster using $\widetilde{POK}1(\beta)$. The challenger appends $\left\langle \boxed{g_0^{\hat{z}}}, \boxed{\hat{c}}, \mathbf{B}_j, \boxed{\hat{\pi}_1}, \boxed{\hat{\pi}_2} \right\rangle$ to AllVotes.

8 **Heads: Generate panic password**. $\mathcal{A}$ is provided $\rho_{(\mathcal{A},j)} = \rho^*$ as in Game 0.

9 **Tails: Give real password**. $\mathcal{A}$ is provided $\rho_{(\mathcal{A},j)} = \rho_j$ as in Game 0.

13 **PreTally**. PreTally proceeds as in Game 0.

We now consider whether $\mathcal{A}$ can distinguish is he is playing Game 0 or Game 1. If $b = 1$, then the games are identical and he will have no advantage. For this reason, the advantage we compute should be scaled by a factor of $1/2$, however since we will be dealing with negligible quantities, we omit this. If $b = 0$, $\mathcal{A}$ is asked to distinguish $\left\langle g_0^{\rho_j}, \mathsf{ReRand}(c_j), \mathbf{B}_j, \pi_1, \pi_2 \right\rangle$ from $\left\langle \boxed{g_0^{\hat{z}}}, \boxed{\hat{c}}, \mathbf{B}_j, \boxed{\hat{\pi}_1}, \boxed{\hat{\pi}_2} \right\rangle$. Since votes are appended, $\mathcal{A}$ can find this submission when $b = 0$.

We first consider if $\mathcal{A}$ can distinguish $g_0^{\hat{z}}$ for a random $\hat{z}$ from $g_0^{\rho_j}$. The adversary can see an encryption of ρ_j, $(c_{(1,j)}, c_{(2,j)}) = \mathsf{Enc}(g_0^{\rho_j}, r)$, from the coerced voter's entry on the Roster.

[2]Note that this is a necessary but not sufficient condition for the submission being a second vote from the coerced voter. There is no restriction on other voters submitting votes using the coerced voter's Roster entry. These will be eliminated.

In Game 0, the following values form a Diffie-Hellman tuple: $\left\langle g, g^r = c_{(1,j)}, y, y^r = c_{(2,j)}/g_0^{\rho_j} \right\rangle$. In Game 1, the last element of this tuple is masked by z and is distributed randomly in $\mathbb{G}_q$. Thus distinguishing them is an example of the decisional Diffie-Hellman problem.

Toward a contradiction, let A^{G1} be an efficient algorithm for distinguishing Game 1 from Game 0 based on this potentially distinguishing factor (we deal with three others next). We can use A^{G1} to gain a non-negligible advantage at the DDH game in the underlying group. The adversary receives challenge tuple $\langle a_1, a_2, a_3, a_4 \rangle$ where $a_1^x = a_2$ for some x. $\mathcal{A}$ must decide if $a_3^x = a_4$. $\mathcal{A}$ computes a random exponent $\omega \leftarrow_r \mathbb{Z}_q$, sets $g = a_1$, sets $e = a_3$, replaces RosterEntry with $c'_j = \langle a_2, a_4 * g_0^{\omega} \rangle$, and places g_0^{ω} in the submitted vote. He submits this transcript to A^{G1}. Upon output Game 0, $\mathcal{A}$ guesses it is a Diffie-Hellman tuple and upon output Game 1, he guesses it is not. Let ϵ_{ddh} be the adversary's advantage at the DDH game.

Next, $\mathcal{A}$ again obtains the coerced voter's entry, c_j, from the Roster. Now he guesses if the submitted vote contains a rerandomization of c_j or not. Let the submitted value, either $\mathsf{ReRand}(c_j)$ or $\hat{c}$, be c^*.

Toward a contradiction, let A^{G2} be an efficient algorithm for distinguishing Game 1 from Game 0 based on this potentially distinguishing factor. We can use A^{G2} to gain a non-negligible advantage at the CPA game of the underlying encryption. $\mathcal{A}$ submits two random messages, $m_1, m_2 \leftarrow_r \mathbb{G}_q$ and receives challenge ciphertext c_{m_b}. The adversary takes a transcript of a game, replaces voter j's entry of Roster with $\mathsf{Enc}_e(m_1, r)$ and replaces c^* with c_{m_b}. $\mathcal{A}$ submits this to A^{G1} and upon a guess of Game 0, $\mathcal{A}$ guess m_0. Let ϵ_{cpa} be the adversary's advantage at the CPA game.

There are two other values in the transcript of the Game that are functionally dependent on $\hat{z}$. First, π_1 will be constructed the same in both games, but will rely on different witnesses. We assumed an idealized proof of knowledge, and recall that we assumed it is witness hiding. Let we bound the information $\mathcal{A}$ can gain from π_1 about the witness at ϵ_{wh-pok}. Second, the submission contains a transcript π_2 that proves knowledge of $\hat{z}$ in $g_0^{\hat{z}}$. We assume $\mathcal{A}$'s advantage at extracting $\hat{z}$ from this transcript is ϵ_{wh-pok}.

By the triangle inequality, $|S_1 - S_0| \leq \epsilon_{cpa} + \epsilon_{wh-pok} + \epsilon_{ddh} + \epsilon_{wh-pok}$

Game 2. We now describe Game 2 and show that $|S_2 - S_1| \leq \epsilon_{cpa}$

6 **Flip coin**.

7 **Heads: Cast ballot**. As in Game 1.

8 **Heads:** $\boxed{\textbf{Give real password}}$. $\mathcal{A}$ is provided $\rho_{(\mathcal{A},j)} = \boxed{\rho_j}$ instead of ρ^*.

9 **Tails: Give real password**. As in Game 1.

We now consider whether $\mathcal{A}$ can distinguish is he is playing Game 1 or Game 2. As before, if $b = 1$, then the games are identical and he will have no advantage. If $b = 0$, $\mathcal{A}$ is asked to distinguish ρ_j from ρ^* given an encryption of ρ_j from Roster. Distinguishing the games provides an advantage in the CPA game following almost the identical reduction already used in Game 1.

Toward a contradiction, let A^{G3} be an efficient algorithm for distinguishing Game 2 from Game 1 based on this potentially distinguishing factor. We can use A^{G3} to gain a non-negligible advantage at the CPA game of the underlying encryption. $\mathcal{A}$ submits the messages, m_0, m_1 and receives challenge ciphertext c_{m_b}. The adversary replaces the coerced voter's entry on Roster with c_{m_b} and uses m_1 as the provided value. He submits this to A^{G3}. Upon output Game 1, $\mathcal{A}$ guesses m_0 (since in Game 1, the roster and provided value are different) and upon Game 2, it guesses m_1 (since in Game 2, the values are the same).

Game 3. We describe Game 3 and show that $|S_3 - S_2| \leq \epsilon_{cpa} + \epsilon_{wh-pok} + \epsilon_{ddh} + \epsilon_{wh-pok}$ and that $S_3 = \mathbf{Pr}[\mathbf{Exp}^{\text{cr-ideal}}_{ES,\mathcal{A}'}(\cdot) = 1]$.

10 **Honest voters vote**. For each voter remaining in U, the challenger posts a ballot following the procedure used in Game 2 for the coerced voter, except the commitments and encryptions do not match. The output to $\mathsf{AllVotes}$ for each voter is $\left\langle \boxed{g_0^{\hat{z}_1}}, \boxed{\mathsf{Enc}_e(g_0^{\hat{z}_2}, r)}, \mathbf{B}_j, \boxed{\hat{\pi}_1}, \boxed{\hat{\pi}_2} \right\rangle$ for a random values $\hat{z}_1$ and $\hat{z}_2$. If the vote is chosen to be not valid, the simulated submission is modified in the same way (*e.g.,* invalid proof). If the vote is a duplicate, the same $\hat{z}_1$ value is used. If the vote is not supposed to match the Roster, this is noted by the challenger.

11 **Corrupt voters vote**. Same as in Game 2.

12 **Coerced voter votes**. Same as in Game 2.

13 **PreTally**. The challenger operates the $\mathsf{PreTally}$ protocol as in Game 2 with some modifications. Since none of the honest votes will having matching $g_0^{\hat{z}_1}$ and $\mathsf{Enc}_e(g_0^{\hat{z}_2}, r)$ values, the challenger will track the votes through the protocol. The challenger eliminates submissions with invalid proofs or duplicate committed passwords as in Game 2, and outputs $\mathsf{UniqueVotes}$. Instead of querying $\widetilde{MIX}(\mathsf{UniqueVotes}, k_2)$, the challenger simulates a mix (as a reencryption mix, it does not depend on $d_{\mathcal{T}}$) and remembers where each honest vote ends up. For each corrupted or coerced tuple, it submits the first two entries to $\widetilde{PET}$. If the test is negative, the entry is removed. For honest votes, it simulates a positive or negative output of $\widetilde{PET}$ according to whether the submission was intended to match the Roster. The remaining list $\mathsf{ValidVotes}$ is output to $\mathcal{A}$ along with each preceding list.

In Game 1, we have seen an argument for why replacing commitments to passwords with a random $g_0^{\hat{z}_1}$ and rerandomized roster entries with a random $\mathsf{Enc}_e(g_0^{\hat{z}_2}, r)$ cannot be distinguished with advantage greater than $\epsilon_{ddh} + \epsilon_{cpa}$. The rest of the modifications consist of replacing queries to the ideal components with simulations. This could be detected if the witnesses could be extracted from the real proofs in Game 2 as opposed to the simulated proofs in Game 3. We denote this advantage as $\epsilon_{mix} + \epsilon_{pet}$ for the mixing and plaintext equality tests.

Since AllVotes, ProvedVotes, UniqueVotes, and ValidVotes contains only random values in each vote submission other than the adversary's own submissions (through corrupt voters and/or the coerced voter), the only useful information it provides is the final output,ValidVotes, and the relative size of each list, Γ. This is exactly the output of $\mathbf{Exp}^{\text{cr-ideal}}_{ES,\mathcal{A}'}$.

Putting everything together,

$$\begin{aligned} |S_3 - S_0| &= |\mathbf{Pr}[\mathbf{Exp}^{\text{cr-ideal}}_{ES,\mathcal{A}'}(\cdot) = 1] - \mathbf{Pr}[\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}(\cdot) = 1]| \\ &= \epsilon_{cpa} + \epsilon_{wh-pok} + \epsilon_{ddh} \\ &= \mathbf{adv}^{\text{cr}}_{ES,\mathcal{A}} \end{aligned}$$

D.2.4 Case 2: fixed β

We show that when β is constant (*e.g.,* 5 or 100), $\mathbf{adv}^{\text{cr}}_{ES,\mathcal{A}} < \delta$, where δ is small but non-negligible. Recall there are V_2 votes with valid proofs and R entries on the ElectionRoster. Each voter submits a vote within an anonymity set. Assume the members of this set are chosen with uniform randomness. The expected number of votes cast in which the coerced voter j is in one or more anonymity set can be described with a binomial distribution.

When $b = 0$, at least one vote will include j in an anonymity set because the coerced voter must include herself in the vote to be valid. The additional inclusions are distributed randomly. When $b = 1$, all the inclusions are distributed randomly. Thus, there is a measurable difference between these distributions, which we call δ. Let $\mathbf{F}(k; p, n)$ be the cumulative distribution function of a Binomial distribution with n trials, p success probability, and k successes. The adversary's advantage is δ,

$$\delta = \frac{1}{2}(\mathbf{F}(\frac{\beta V_2}{R}; V_2, \frac{\beta}{R}) + 1 - \mathbf{F}(\frac{\beta V_2}{R} - 1; V_2 - 1, \frac{\beta}{R})).$$

While we could perhaps provide a bound on this value, we are ultimately uninterested in Case 2 and are considering it only to motivate Case 3. Therefore we simply note it is

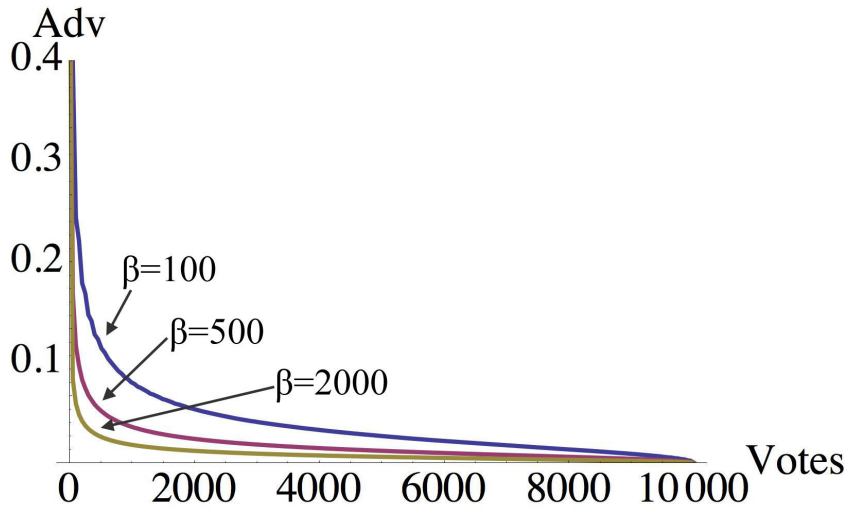


Figure D.1: Adversary's advantage over one half (δ) versus β and votes cast (V_2) in Case 2.

a non-negligible amount, and provide in Figure D.1 some values of δ plotted against the number of cast votes and β.

In Case 1, we made assumptions about adversarial uncertainty with respect to certain voting behaviours. For example, the total number of votes cast will always differ by one between $b = 0$ and $b = 1$ but it also differs according to $\mathcal{D}$. (Similarly the number of votes in the final output will as well, but in $\mathbf{Exp}^{\text{cr-ideal}}_{ES,\mathcal{A}'}$, the number of votes in the final output is also provided to $\mathcal{A}$.) Unfortunately, the anonymity set membership distribution cannot be described within adversarial uncertainty: in $\mathbf{Exp}^{\text{cr-ideal}}_{ES,\mathcal{A}'}$, $\mathcal{A}$ does not have access to any information posted to $\mathsf{AllVotes}$ and the protocol clearly provides an expected value for j's inclusion in an anonymity set. Therefore, we conclude Case 2 is not secure.

Game 0-weak. We consider a modified experiment, $\mathbf{Exp}^{\text{cr-weak}}_{ES,\mathcal{A}}$, for which $\mathbf{adv}^{\text{cr-weak}}_{ES,\mathcal{A}}$ is negligible for Case 2. We do this to clearly illustrate what blocks the proof from going through. We hope this makes the motivation for and solution in Case 3 easier to understand. We illustrate this through a game: Game 0-weak.

6 **Flip coin**.

7 **Heads: Cast ballot**. As in Game 0.

8 **Heads: Give panic password**. As in Game 0.

9 **Tails: Give real password**. As in Game 0.

10 **Honest voters vote**. Iff $b = 1$, the challenger will select for the first honest voter an anonymity set that includes voter j. The other members of the set are chosen randomly, and the process for the rest of the honest voters proceeds as in the original Game 0 (with randomly selected, β-sized anonymity sets).

Game 0-weak is too strong of a modification to consider Case 2 coercion resistant; however were it made, the security would follow as in Case 1 (with this modification preserved through the sequence of games).

D.2.5 Case 3: stealth votes

We consider the case where β is required to be at least a constant value (*e.g.,* 5 or 100) but voters can submit stealth votes where $\beta = R$. We show that if a coerced voter's coercion-resistant strategy is to submit their real vote as a stealth vote, $\mathbf{adv}^{\mathrm{cr}}_{ES,\mathcal{A}}$ is negligible. We do make one small change to $\mathbf{Exp}^{\mathrm{cr}}_{ES,\mathcal{A}}$: instead of the coerced voter's real vote being appended to the cast ballots, it is inserted at a random place (*i.e.,* she votes her real ballot at some arbitrary time after being coerced). If we used Game 0 directly, the use of appends means that the coerced voter's vote will always be first (when $b = 0$). $\mathcal{A}$ can simply inspect this vote and see if it is a stealth vote. It will always be when $b = 0$ and will only be so with non-certain probability when $b = 1$. We believe this gives the adversary unnecessary power. We describe our change as Game 0-iii and then argue for its validity.

Game 0-iii.

6 **Flip coin**.

7 **Heads: Cast ballot**. If $b = 0$, the challenger constructs a ballot for coerced voter j using password ρ_j as in Game 0. Likewise, the challenger constructs a ballot of the form $\langle g_0^{\rho_j}, \mathsf{ReRand}(c_j), \mathbf{B}_j, \pi_1, \pi_2 \rangle$ where π_1 is the output of $\widetilde{POK1}(\beta)$ and π_2 is the output of $\widetilde{POK2}$. In Game 0, due to the use of an append operation, the adversary knows exactly where this ballot, if it exists, will be posted. In this game, the challenger holds the ballot until he posts the votes from the honest voters and at that time, he posts this ballot as well in a random location.

8 **Heads: Give real password**. As in Game 0.

9 **Tails: Give real password**. As in Game 0.

Given the adversary is passive and must corrupt voters at the beginning of the protocol, we believe that Game 0 gives the adversary too much power. This is not a problem until we have a protocol where the security should hold under reasonable assumptions but the proof will not go through, as is the case here. We propose Game 0-iii as reasonable modification to $\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}$ that preserves the security of the system against a *passive* adversary, who must corrupt voters at the beginning of the protocol. With a non-adaptive adversary, the coerced voter knows they are being coerced and in the $b = 0$ case, they are attempting to deceive the adversary. Game 0-iii simply adds another dimension to the deception.

In the real-world, the original $\mathbf{Exp}^{\text{cr}}_{ES,\mathcal{A}}$ does not model the intended strategy for a coerced voter. If a coercer demands a voter's password, the voter cannot cast a real vote immediately because they are in the presence of the adversary. It is likely they may have already voted with their real password, or will wait some period of time before submitting. In other words, the timing of when they submit their real vote is invariant to the time that they are coerced, unless if they happen to coincide.

If we accept Game 0-iii as a definition of coercion resistance against a passive adversary, then security of Case 3 follows from the negligible distinguishability between Game 0-iii and a similarly adjusted Game 1-iii and on through the same steps we used in Case 1. If we consider this potentially distinguishing feature that not all votes have anonymity sets of the full size in isolation (Case 1 proof covers the others), we must introduce a new assumption: we assume that other voters are submitting stealth votes (because they are privacy sensitive) and the adversary has no reliable expected value for the quantity of stealth votes in the election. To accomplish this, we expand $\mathcal{D}$ to include the distribution of stealth and non-stealth votes, and as with the other aspects of this distribution, we assume adversarial uncertainty.

D.3 Performance Comparison

In this section, we provide some additional details of the comparison we performed in Section 11.6. For reference, we duplicate the table provided in the main body here as Table D.1.

Selections and Civitas both use Elgamal encryption, while JCJ and AFT use a modified variant of Elgamal that adds one modular exponentiation to encryption, re-randomization, and decryption. For performance comparison, we use Elgamal. Elgamal requires 2 modular exponentiations for encryption, 2 for reencryption, and 1 for decryption. Proof of knowledge of a plaintext requires 3 exponentiations to compute and 4 to verify.

Selections, JCJ, and AFT use threshold decryption while Civitas uses distributed decryption. For streamlining tasks involving all trustees with ones involving an authorized

		Civitas	AFT	Selections
Registration	Registrar	7	9	2α
	Voter	11	10	4α-1
Casting	Voter	10	24	$(2\beta + 9)$
Pre-Tally	Check Proofs	$4V_0$	$20V_0$	$(4\beta + 6)V_0$
	Remove Duplicates	$(1/2)(V_1^2 - V_1)(8T + 1)$	—	—
	Check Removal	$(1/2)(V_1^2 - V_1)(8T + 1)$	—	—
	Mix	$8V_2T + 4RT$	$20V_2T$	$12V_2T$
	Check Mix	$4V_2T + 2RT$	$10V_2T$	$6V_2T$
	Remove Unregistered	$(8A + 1)V_2R$	$(16T + 8)V_2$	$(8T + 1)V_2$
	Check Removal	$(8A + 1)V_2R$	$(16T + 10)V_2$	$(8T + 1)V_2$

Table D.1: Comparison of the efficiency of the main protocols in Civitas, AFT, and Selections, measured with modular exponentiations.

set, we assume all trustees are needed. Thus we use distributed decryption for performance comparison. In addition to the 1 modular exponentiation for plain decryption, we need an additional $3T$ where T are the number of trustees. To verify that the decryption was done correctly requires $4T + 1$ exponentiations.

Part of the distributed decryption process involves each trustee proving that a tuple of values is a Diffie-Hellmen tuple. With Chaum-Pedersen, this is 2 exponentiations to compute and 4 to verify. This proof can also prove values are rerandomized correctly. The knowledge of discrete log, with Schnorr, is 1 to compute and 2 to verify. Knowledge of a representation requires 1 to compute and 3 to verify. For one of the proofs during casting, AFT requires knowledge of a representation with 3 generators. This requires 1 to compute and 4 to verify.

ZKPs can be made 1-out-of-m by scaling both the computing and verifying requirements by m. ZKPs can be made designated verifier by adding 1 exponentiation to both computing and verifying (if the verifier's key is known in advance). We assume all commitments are based on hash functions, with the exception of the Chameleon commitment used in a designated verifier proof. We assume this commitment is a Pedersen commitment. Even though Pedersen commitments contain two exponents, they are cheaper than two modular exponentiations to compute using standard optimizations for multi-exponentiations. They are sometimes counted as 4/3 exponentiations; for simplicity, we assume they require a single exponentiation.

A plaintext equality test requires each of the T trustees to blind both ciphertext components: $2T$ exponentiations; compute a Chaum-Pedersen proof: $2T$; and perform a distributed decryption: $4T$+1. This totals $8T + 1$ for computing the test and $8T + 1$ to verify it was performed correctly.

Finally, a mix-network requires each of the T trustees to reencrypt each of N ciphertexts

in the input: $2N$ exponentiations. With randomized partial checking, each trustee actual performs two mixes: increasing the workload to $4N$. Finally, the input is not a list of single ciphertexts but rather tuples that include multiple ciphertexts: each of which has to be individually reencrypted. If there are N tuples with τ ciphertexts in each tuple, mixing costs $4NT\tau$. With randomized partial checking, half of the recencryptions have their randomization revealed. Checking these requires half the cost of mixing: $2NT\tau$.

D.3.1 Civitas

During registration, the registrar computes an encryption (2) and proves knowledge (3). It then computes a second encryption of the same value (2) and proves it is a rerandomization. This second proof is designated verifier (2+1). The voter is supplied with the plaintext and randomization for the second value. She checks the ciphertext is correct through encryption (2), verifies the knowledge of plaintext (4), and verifies the proof of randomization (4+1).

Vote casting consists of two encryptions (4) and a proof of simultaneous knowledge (6) of the plaintexts.

Tallying begins by checking each proof ($4V_0$). Then PETs are performed between each pair of submitted votes with correct proofs ($\binom{V_1}{2}(8T+1)$) and checking these PETs (same). The V_1 tuples, with two ciphertexts in each tuple, are mixed ($8V_2T$) and checked ($4V_2T$). The R values on the Roster are also mixed, with one ciphertext in each tuple: mix ($4RT$) and check ($2RT$). Finally, PETs are performed between each pair of entries in R and V_2 ($(8T+1)V_2R$) and checked ($(8T+1)V_2R$).

D.3.2 AFT

For the comparison, we use the 2010 version of AFT [AFT10] (*cf.* [AFT07])

During registration, the registrar forms a credential (3) and proves its validity with respect to two keys using two designated verifier Chaum-Pedersen proofs (6). The voter checks these proofs (10).

Vote casting consists of four encrypted values (8), where two of the plaintexts were computed prior to encryption (2) and one submitted value is computed but not encrypted (1). The voter proves knowledge of the four encryptions (12). The voter also proves a representation (1) that relates an encrypted value to the non-encrypted value.

Tallying begins by checking the plaintext proofs (16) and representation (4). Duplicates can be removed without computing any exponentiations. AFT includes a step where a submitted value is encrypted: we omit this step as the first mix can perform this at no additional cost. The mixing is performed on tuples with five ciphertexts each ($20V_2T$ plus

$10V_2T$ to check). Finally, removing unregistered voters requires, per vote, an exponentiation (1), Chaum-Pedersen to prove matching exponents (2), a PET $(8T+1)$, and then the same three steps a second time. Finally, the two PETs and proofs are checked $(16T+10)$ for each vote left.

D.3.3 Selections

Prior to registration, the voter prepares α encryptions (2α). The registrar rerandomizes each (2α). The voter checks the rerandomization of all but one $(2(\alpha-1))$.

Vote casting consists of a computed value (1), a proof of knowledge of this value (1), a rerandomization (2), a 1-out-of-β proof of rerandomization (2β), an encrypted value (2), and a proof of knowledge of the plaintext for this value (3).

During tallying, the three proofs are checked $(2+4\beta+4)$ for each cast vote. Duplicates are removed without any exponentiations. The mix is of tuples with three ciphertexts ($12V_2T$ plus $8V_2T$ to check). Finally, a single PET is performed for each remaining tuple $(8T+1)$.

D.4 Augmented Transcript Cards

Future work could explore the creation of an augmented transcript card that provides verifiable, bare-handed mechanisms for erasures, pre-committed randomness and a voter commitment to which of the α submissions they will choose. It could include the following features,

- A serial number.
- A commitment to the contents for the Roster prior to the protocol.
- A place for the voter to commit to which ciphertext she will retain prior to the reencryptions being printed. This could be accomplished with a simple hole-punch. The voter performs the hole-punch in front of the registration agent and the card is checked before the voter leaves to ensure the scratched off cell matches the committed spot.This assumes that the machine she interacts with cannot detect the hole.
- Confirmation codes printed under each scratch-off surface for the voter to prove to the registrar she complied with the erasure.
- To eliminate covert channels, randomization is precommitted to.
- The randomization values could be supplied to a one-way function to generate the erasure codes.
- A commitment to the contents of the Roster after the protocol.